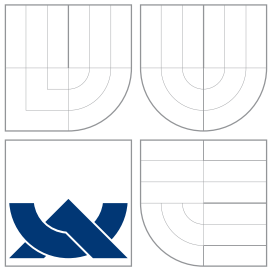
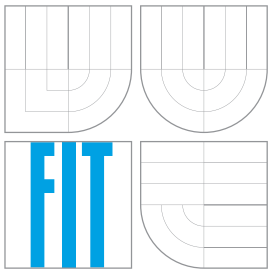


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

DETEKCIA NARUŠENIA POČÍTAČOVEJ SIETE

INTRUSION DETECTION IN COMPUTER NETWORK

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — ANDREJ HANK
AUTHOR

VEDOUCÍ PRÁCE — KOŘENEK JAN, Ing.
SUPERVISOR

BRNO 2007

Zadanie bakalárskej práce

1. Seznamte se s HW a SW architekturou IDS sondy Traffic Scanner implementovanou na platformě COMBO6X.
2. Analyzujte možnosti HW akcelerace stávajících open-source IDS nástrojů.
3. Navrhněte a vytvořte programové vybavení umožňující konfiguraci sondy, sběr dat a jejich analýzu v reálném čase. Pro účely zpětné analýzy podezřelého síťového provozu navrhněte systém ukládání těchto dat a jejich zpřístupnění k off-line zpracování.
4. Analyzujte možnosti integrace a spolupráce IDS sondy Traffic Scanner s ostatními zařízeními pro sledování síťového provozu. Navrhněte koncepci detekce narušení počítačové sítě pomocí IDS sondy Traffic Scanner.
5. Ověřte funkcionalitu navrženého řešení. Proveďte kontrolní měření a srovnejte možnosti čistě SW IDS s hardwarově akcelerovaným IDS pomocí COMBO6X karty.

Licenčná zmluva

Licenčná zmluva je uložená v archíve Fakulty informačných technológií Vysokého učenia technického v Brne.

Abstrakt

Stále rastúce rozšírenie a priepustnosť počitačových sietí prináša mnoho bezpečnostných hrozieb. Jedným z prostriedkov zabezpečenia sú systémy detekcie narušenia (IDS). Nízka priepustnosť softwérových IDS vyúsťuje vo vývoj hardwérových akcelerátorov. Konkrétnym je sonda Traffic Scanner projektu Liberouter využívajúca technológiu FPGA. Jadro akcelerácie spočíva vo vyhľadávaní jednoduchých reťazcov v obsahu paketu. Komplexný popis reťazcov zaručujú regulárne výrazy. Táto práca obohacuje sondu Traffic Scanner o možnosť pokročilého vyhľadávania reťazcov pomocou Perl Compatible Regular Expressions (PCRE) implementovaným transformátorom. Ďalej prináša návrh a implementáciu programového vybavenia umožňujúceho využiť funkcionalitu akcelerátoru užívateľom. Podáva koncepciu detekcie narušenia sieťe s využítím sondy Traffic Scanner a možnosti spolupráce s inými bezpečnostnými zariadeniami.

Klíčová slova

sieťová bezpečnosť, PCRE, regulárne výrazy, konečné automaty, syntaktický analyzátor, IDS, Snort, FPGA, qt4

Abstract

Continuous spreading and growing bandwidth of computer networks brings many security threats. Intrusion Detection System (IDS) is a mean to provide network security. Software IDS aplications gain only low throughput and that is why hardware accelerators are under heavy development. Probe Traffic Scanner is a hardware accelerator developed in Liberouter project with use of FPGA technology. Main core of acceleration is searching packet payload for simple suspicious strings. Regular expressions provide complex way of describing strings. This bachelor thesis adds feature of searching according to Perl Compatible Regular Expressions (PCRE) to Traffic Scanner Probe by implemented transformer. In addition design and implementation of control software allowing users to use functions provided by the Probe have been created. Conception of intrusion detection in network utilizing Traffic Scanner is outlined so as possibilities of cooperation with other security devices.

Keywords

network security, PCRE, regular expressions, finite state machines, parser, IDS, Snort, FPGA, qt4

Citace

Andrej Hank: Detekcia narušenia počítačovej siete, bakalářská práce, Brno, FIT VUT v Brně, 2007

Detekcia narušenia počítačovej siete

Prohlášení

Prehlasujem, že som túto bakalársku prácu vypracoval samostatne pod vedením pána Jana Kořenka.

.....
Andrej Hank
15. května 2007

Poděkování

Chcem poďakovať vedúcemu práce Ing. Jánovi Kořenkovi a Petrovi Kobierskému za pomoc pri smerovaní práce a venovaný čas. Ďakujem tiež kolektívu projektu Liberouter za možnosť realizácie, technickú a informačnú podporu.

Obsah

Kapitola 1

Úvod

Dnešná doba je charakteristická masovým využívaním výpočtovej techniky a počítačových sietí. Žijeme v „informačnej spoločnosti“ a osobný počítač sa stáva nutnosťou a práca s ním každodennou rutinou pre stále väčšiu skupinu obyvateľstva. Stúpa životná úroveň a kúpyschopnosť obyvateľstva i organizácií. Dnešné PC majú dostatočný výkon pre potreby bežného užívateľa, aj pokiaľ sa nejedná o technologickú špičku. Internetové pripojenie je dnes široko dostupné verejnosti najmä vďaka rozmachu technológie ADSL a postupne si ľudia naň zvykajú ako na štandardnú službu akou je napríklad telefonické spojenie. Vo firemnom prostredí je sieťové pripojenie nevyhnutné.

Tento stav, kde nám výpočtová technika uľahčuje život, si však vyberá svoju daň. Ľudia majú množstvo negatívnych vlastností. Radi získavajú statky, ktoré im nepatria. V dnešnej dobe sú to najmä informácie, ktoré môžu mať veľkú cenu.

Táto ľudská podstata sa bohato prejavuje aj vo svete výpočtovej techniky. Vírové hrozby, trójske kone, DoS útoky, phishing sú na každodennom poriadku a spôsobujú nemalé hospodárske škody alebo naopak, niektorým organizáciám stúpajúce príjmy. Odohráva sa tu nekončiaci boj medzi útočníkmi a obrancami. Počet a sofistikovanosť útokov neustále stúpa.

Základným prostriedkom na zabezpečenie sieťovej premávky sú firewally. Pokročilejšie možnosti v tejto oblasti poskytujú systémy detekcie narušenia IDS. Zväčša sú implementované ako softwér bežiaci na obecnom procesore. Najznámejším a najrozšírenejším je zrejme IDS Snort. Tieto systémy obsahujú väčšinou komplexné riešenie detekcie narušenia a sú flexibilné.

Najnáročnejšou úlohou IDS je prehľadávanie veľkého množstva dát pre výskyt nebezpečných reťazcov. Počty reťazcov sa rátajú na tisíce. Najväčšou nevýhodou softwérových riešení IDS je ich nízka priepustnosť. Tento nedostatok je spôsobený užívaním obecného procesoru, ktorý nie je optimalizovaný pre riešenie špecifických operácií prevádzaných IDS aplikáciami. Priepustnosť sa pohybuje v rádoch stoviek Mb.

So stále zvyšujúcou sa priepustnosťou a príchodom 1Gb a 10Gb sietí softwérové IDS nedostačujú. Sú vyvíjané snahy o presun výkonovo najnáročnejších operácií do špecializovaného hardwéru. Existujúce komerčné riešenia, realizované v špecializovanom hardwéry, sú však veľmi finančne nákladné.

Perspektívu v tejto oblasti ukazuje technológia FPGA (Field-Programmable Gate Array). Realizácia potrebných operácií hardwérovej akcelerácie s využitím tejto technológie je predmetom intenzívnych výskumov prinášajúcich dobré výsledky.

Konkrétnym príkladom vyvíjaného hardwérového akcelerátoru IDS Snort je zariadenie Traffic Scanner projektu Liberouter. Prvotné verzie zariadenia, ktorého hlavnou úlohou je

vyhľadávanie jednoduchých reťazcov v obsahu paketov, prinášajú veľmi dobré výsledky. Vyhľadávanie je založené na použití konečných automatov (KA). Pre komplexné pokrytie hardwérovej akcelerácie je potrebné riešiť otázky pokročilého vyhľadávania reťazcov regulárnymi výrazmi. V prípade akcelerovaného IDS Snort sú to PCRE (Perl Compatible Regular Expressions). Toto rozšírenie je možné vďaka použitiu KA. Ďalšou potrebou akcelerátoru je funkčné programové vybavenie sprístupňujúce ovládanie užívateľom. Efektívnu a jednoduchú prácu poskytuje grafické užívateľské prostredie (GUI).

Práca je členená do šiestich kapitol. Nasledujúca kapitola čitateľa uvádza do problematiky sieťovej bezpečnosti, poskytuje prehľad techník sieťových útokov a pomerne obsiahly popis prostriedkov pre zabezpečenie. Pozornosť je venovaná najmä prostriedkom IDS a ich rozličným druhom. Podrobne popísaný je aj hardwérový akcelerátor Traffic Scanner a teória konečných automatov. Tretia kapitola vysvetľuje proces vytvárania akcelerátora Traffic Scanner, najmä transformáciu PCRE do kódu VHDL. Štvrtá kapitola popisuje softwérovú vrstvu sondy a vytvorené programové vybavenie. Predposledná kapitola zhodnocuje sondu z pohľadu funkčného prvku v sieťovej infraštruktúre, podáva užívateľskú koncepciu a víziu spolupráce s inými bezpečnostnými zariadeniami. Na záver je celá práca zhrnutá a zhodnotená a sú podané smery ďalšieho vývoja.

Kapitola 2

Teoretický rozbor

2.1 Útoky

Pokusy o narušenie bezpečnosti systémov nazývame útokmi. Väčšinou je ich cieľom priniesť útočníkovi profit, alebo poškodiť niekoho iného, či už konkrétnu obeť alebo anonymnú osobu. Aj toto poškodenie môže priniesť v konečnom dôsledku útočníkovi zisk.

Pre prevedenie útokov sa využíva dobre známych zaužívaných techník, ale neustála invencia útočníkov prináša techniky nové, sofistikovanejšie, využívajúce chyby novo-používaných technológií.

2.1.1 Oblasti útokov

Každý útok sa zameriava na zneužitie v určitej oblasti napádaného systému podľa techniky, ktorú útok využíva. Aj keď výsledok rôznych útokov je rovnaký, môžu pre dosiahnutie svojho cieľa využívať rozdielnu oblasť. Hlavnými oblasťami zamerania útokov sú:

Dôvera – útočník získa „vzájomnú dôveru“ u určitej služby, programu. Výsledkom je získaný prístup bez hesla. Takéto zneužitie je ťažko odhaliteľné.

Integrita – útočník pozmení dôležité konfiguračné súbory, alebo spustiteľné programy a tým si vytvorí „zadné vrátka“ do systému.

Dostupnosť – dostupnosť obete je obmedzená a služby nie sú plne využívané.

Kontrola – útok s cieľom prevziať úplnú kontrolu nad systémom s administrátorskými právami.

2.1.2 Typy útokov

V tejto sekcii si popíšeme najčastejšie používané typy útokov a ich princípy. Znalosti v tejto oblasti sú nevyhnutné pre uvedomenie si rizík, proti ktorým sa treba brániť a s ktorými treba rátať pri návrhu a použití bezpečnostných systémov.

Medzi najznámejšie a najčastejšie používané útoky patria:

Skenovanie portov – býva zväčša predvojom pre skutočné útoky. Má za úlohu zistiť, ktoré porty sú na vyhliadnutej obeti otvorené, aké služby na nich bežia, konfiguráciu firewallu a ďalšie podrobnosti týkajúce sa obeti. Techniky skenovania portov zväčša

využívajú chybové stavy protokolu TCP a následné zasielanie oznámení o nich protokolom ICMP.

Programy ako *nmap* sú schopné odhadnúť i používaný operačný systém na základe jeho *fingerprintu* (súbor jedinečných znakov). Táto technika je založená na fakte, že operačné systémy sa líšia v detailoch implementácie štandardných protokolov, napríklad vo veľkosti počiatočného TCP okna, sekvenčného čísla ACK, TCP flagoch alebo fragmentácii. Štandardné správy o verziách bežiacich služieb (ftpd, pop3...) z „uvítacích správ“ nie sú dôveryhodné, pretože môžu byť často zmanipulované kvôli ochrane. Tieto informácie sú pre útočníka veľmi cenné, pretože akonáhle pozná systém a jeho verziu, na ktorú chce útočiť, môže využiť už známe exploity.

DoS – Denial of Service – sú útoky zamerané na odstavenie bežiacej služby. DDoS je distribuovaná forma DoS, kedy sa na útoku podieľa viac, väčšinou infikovaných útočníkov. Podľa techniky, ktorú DoS útoky používajú rozlišujeme:

ICMP flooding – host odpovedá na ICMP echo request paketom ICMP echo reply. Obeť „zaplavíme“ veľkým množstvom týchto paketov a odpovedanie na ne ju stojí zdroje. Adresa útočníka je zväčša zmenená, takže odpovede dostáva vždy niekto iný.

SYN flooding – využíva neukončenie trojfázového TCP handshake (ustanovenie spojenia), fronta neustanovených spojení tak neustále narastá a zaťažuje obeť.

Land – obeti sú posielané pakety obsahujúce rovnakú zdrojovú a cieľovú IP adresu. Komunikácia sa *zacyklí*.

Teardrop – využíva prekrývanie fragmentov pokiaľ je datagram väčší ako MTU (Maximum Transmission Unit). Môže spôsobiť pretečenie vyrovnávacej pamäte.

Používajú sa i ďalšie druhy útokov ako UDP flooding, Ping of Death alebo Smurf.

Exploity – týmto názvom sa označujú dobre známe chyby v konkrétnych službách a aplikáciách. Väčšinou sú zdokumentované priamo výrobcom. Vyznačujú sa použitím špecifického reťazca.

Malware – všeobecne označuje škodlivý software. Do tejto kategórie patria víry, trójske kone, červy, sypware a adware. Pre infiltráciu do systému obete využívajú dôverčivosť užívateľa, zvyčajne bývajú zamaskované a pôsobia neškodne. Väčšinou sa vyskytujú na internetových stránkach s pochybným obsahom. Iný malware, ako napríklad červy, sa aktívne snaží zneužiť známy exploit prostredníctvom siete, útočí na všetky dostupné počítače a šíri sa lavínovite. Dopady malware sú veľmi rôzne čo sa týka ich závažnosti - od zobrazovania nevyžiadanej reklamy po stratu alebo zneužitie významných dát. Poskytovanie zabezpečenia v tejto oblasti je už dlhší čas široko rozvinuté a využíva hľadanie vzoriek dát charakteristických pre daný malware v súborovom systéme.

Spoofing a phishing – všeobecne využívajú zmenu identity. Phishing spôsobuje presmerovanie užívateľa na dokonalú napodobeninu významnej internetovej stránky (privlastňuje si jej identitu) s cieľom zneužitia jeho osobných údajov. Zvyčajne sa jedná o bankové inštitúcie. Útok býva inicializovaný podvodným formálnym emailom nabádajúcim k návšteve zamaskovanej stránky.

2.2 Prostriedky zabezpečenia

Pre odvrátenie hrozieb útokov a celkové zabezpečenie systémov sa v súčasnosti používajú rôzne typy zariadení pričom každý druh je niečim špecifický, určený pre istý účel. Ďalšie rozdiely sú najmä v komplexnosti riešenia bezpečnostnej otázky. Vyskytujú sa v softwérovej i hardwérovej realizácii.

2.2.1 Firewall

Základným bezpečnostným prostriedkom je firewall s počiatkami v roku 1988. Kontroluje sieťovú premávku medzi zónami s rôznym stupňom dôvery (Internet – LAN). Je základnou súčasťou priamo podporovanou operačným systémom. Firewall koná podľa predom definovaných pravidiel, ktoré rozhodujú na základe analýzy hlavičky IP datagramov – zdrojovej a cieľovej adresy, čísiel portov a použitého protokolu. Pracuje teda na sieťovej vrstve.

Štandardnou vlastnosťou firewallov je *stavové filtrovanie*, keď firewall udržuje záznamy o stave všetkých relácií a filtrovacie pravidlá sa používajú na základe stavu konkrétnej relácie.

Ďalšou generáciou firewallov sú tzv. „firewally aplikačnej vrstvy“. Takéto zariadenie rozumie aplikačným protokolom a vie detegovať ich zneužitie. Táto technológia sa označuje aj ako „deep packet inspection“.

Firewally sú často využívané aj sofistikovanejšímy bezpečnostnými zariadeniami ako IPS, pre podniknutie protiopatrení voči zistenému útoku.

2.2.2 Honey pot

Sú to programy, ktoré simulujú bežiace sieťové služby. Slúžia ako návnada pre útočníka, ktorý chce zneužiť zraniteľné služby. Sieťová premávka je sledovaná, procesy systému sú logované a poskytujú informácie o útočníkovej aktivite a dierach v systéme. Ďalšie využitie je ako ochrana dôležitých serverov, kedy je útočník z portu obete presmerovaný do pasce honey potu. Po detekcii útoku nasledujú adekvátne protiopatrenia.

Keďže služby bežiace na takomto systéme nie sú skutočne využívané užívateľmi, každé spojenie je podozrivé. Honey pot je často zraniteľný rovnako ako skutočné obete, a tak sa môže stať, že útočník prevezme nad systémom kontrolu a zneužije ho. Preto je dôležité zaobchádzanie so systémovým logom, najlepšie jeho skladovanie mimo samotného systému. Sám systém musí byť správne nastavený a pozorne sledovaný, či uňho nedošlo k zneužitiu.

2.2.3 Padded Cell Systems

Sú systémy účelom zhodné a technológiou veľmi podobné honey potom. Väčšinou spolupracujú so zariadeniami IDS, tie okamžite detekovaný útoku oznámia a útočník je presmerovaný do pasce. Zásadný rozdiel je však vtom, že služby ako aj celé systémové prostredie je simulované. Preto útočník nemôže spôsobiť žiadnu skutočnú škodu. Prostredie by malo byť simulované čo najvernejšie, aby útočník systém neodhalil.

2.2.4 IDS

Intrusion Detection System (systém detekcie narušení) získava informácie o sledovanom systéme. Jeho účelom je sledovať bezpečnostné prieniky, pokusy o ne, zistiť systémové zraniteľnosti, ktoré by k prienikom mohli viesť. V súvislosti s týmito systémami sa vyskytujú

pojmy „false negative“, značí situáciu keď IDS nezachytí útok alebo pokus oň, a „false positive“, značí situáciu pokiaľ IDS označí za útok neškodnú akciu.

IDS systémov existuje značné množstvo. V tejto sekcii sú bližšie vysvetlené všeobecné pojmy a charakteristiky a taktiež rozdelenie týchto systémov podľa základných princípov. Taktiež sú tu vysvetlené odlišnosti a špecifiká jednotlivých typov, ich výhody a tiež nedostatky. Tejto kategórii prostriedkov zabezpečenia bude venovaná najväčšia pozornosť s ohľadom na zameranie tejto bakalárskej práce.

IDS zariadenia pre svoju prácu využívajú tieto základné zdroje informácií:

- **knowledge base** – súbor známych útokov, ich signatúr (CVE) [11].
- **konfigurácia systému** – podáva informácie o stave systému.
- **audit** – informácie poskytované systémom o vnútornej práci a správaní systému. V UNIXovom prostredí ide o *syslog* a *„accountig“* – sledovanie užívateľov, využívania zdrojov systému. V prostredí Windows sa jedná o *Event log*.

Efektívnosť IDS určujeme nasledujúcimi kritériami:

Presnosť – parameter sa odvíja od správnej detekcie útokov, a správneho označenia, že o útok sa nejedná. Nepresnosť značí, že IDS označí legitímnu akciu ako útočnú alebo podozrivú (false positive).

Výkon – značí rýchlosť spracovávania informácií. Pokiaľ IDS nie je dostatočne výkonný, klasifikácia a teda aj následná ochrana v skutočnom čase nie je možná.

Úplnosť – je vlastnosť IDS detegovať všetky útoky. Nekompletnosť značí, že IDS neobjaví skutočný útok (false negative). Tento parameter je ťažké vyhodnotiť, pretože nie všetky útoky sú známe a odhaliteľné pomocou bázy znalostí.

Bezporuchovosť – aj samotné IDS sú terčom útokov, najčastejšie sú DoS útoky. Bezporuchovosť udáva mieru systému odolnosti proti týmto útokom. Pokiaľ IDS beží nad štandardným operačným systémom, je minimálne takisto zraniteľný ako samotný operačný systém.

Dochvíľnosť – určuje rýchlosť reakcie a rozšírenie informácie o útoku s cieľom o ňom informovať a/alebo podniknúť protiopatrenia. Je to významný parameter, pretože pri pomalej rýchlosti reakcie môže útočník s vyššou pravdepodobnosťou podvrhnúť auditné informácie – zamaskovať útok.

Knowledge vs. behavior based IDS

Podľa základného rozdelenia založeného na detekčnej metóde rozlišujeme dva komplementárne smery IDS zariadení - založené na správaní (*behavior based*) a založené na báze znalostí o útokoch (*knowledge based*).

Knowledge based IDS: Systémy tohoto druhu sú založené na dlhodobo akumulovanej báze znalostí o útokoch a systémových zraniteľnostiach. IDS využíva tieto informácie pre detegovanie týchto útokov a zneužitie zraniteľností. Je tu uplatnený rovnaký princíp ako v antivírových programoch – v dátach sú vyhľadávané vzorky špecifické

pre určité víry. Zariadenie vyhľadáva vzorky dát špecifické pre útoky z bázy znalostí. Akcia, ktorá nie je považovaná za útok, je považovaná za neškodnú. Presnosť tejto metódy je vysoká, ale úplnosť závisí na aktuálnosti bázy znalostí, ktorá sa musí pravidelne aktualizovať.

Výhodou je teoreticky nízka miera falošných poplachov (false positive) a kontextuálna analýza útoku je pre administrátora jednoduchá, preto môže efektívne zasiahnuť.

Hlavným nedostatkom je udržovanie aktuálnosti bázy znalostí. Podľa už spomenutej paralely s antivírovými programami sa môžu vyskytnúť útoky na tieto systémy s cieľom nedovoliť IDS aktualizáciu bázy znalostí a znemožniť tak detekciu útoku.

Útoky sú úzko zviazané s operačným systémom, aplikáciami a ich verziami a teda aj IDS systém je úzko spätý s týmto prostredím a musí čeliť otázkam generalizácie. Ďalším závažným nedostatkom je detekcia útokov prichádzajúcich zvnútra – napr. zneužitie privilégií. Takýto útok je ťažko odhaliteľný pretože nevyužíva žiadnu známu zraniteľnosť.

Behavior based IDS: Hlavnou myšlienkou týchto systémov je odhalenie útoku pozorovaním odchýlky od bežného očakávaného správania. Aktivita systému sa porovnáva s referenčným modelom, ktorý musí byť v počiatku vytvorený. Zväčša býva založený na štatistikách – priemerných hodnotách dôležitých ukazateľov a ich odchýlok (počet otvorení významných súborov, počet spustení príkazu, využitie zdrojov). Po vytvorení modelu (databáze) systém sleduje aktivitu systému, porovnáva výsledky s referenčným modelom a model sa vyvíja. Pokiaľ deteguje odchýlky, vyskytla sa anomália. Všetko čo sa vymyká vopred naučenému chovaniu je považované za odchýlku, a teda generuje poplach.

Najväčšou výhodou je detekcia ešte neznámych útokov, pretože s najväčšou pravdepodobnosťou budú vykazovať odchýlku od normálneho správania. Preto je úplnosť vysoká. Naopak rastie nepresnosť – normálne používanie môže byť často považované za útok. Výhodou je aj detekcia útokov zvnútra, ktoré sú systémami založenými na báze znalostí len veľmi obtiažne odhaliteľné – zneužitie privilégií.

Najväčšie riziko sa vyskytuje pri vytváraní referenčnej databázy. Pri tomto úkone systém musí byť „čistý“, nie napadnutý, pretože inak sa toto chovanie bude považovať za normálne a nebude vykazované ako anomália. Tento princíp je značne výpočtovo náročný. Podobný princíp využívajú aj neurónové siete, taktiež nesú rovnaké výhody a riziká.

Niektoré *expertné systémy* kombinujú oba vyššie uvedené prístupy. Obsahujú množinu pravidiel, ktoré popisujú útoky (v tvare AK podmienka POTOM dôsledok). Auditné informácie sú preložené na fakty, ktoré nesú sémantický význam. Odvodzovacie jadro systému vytvára úsudky podľa týchto faktov a pravidiel. Popritom vytvárajú štatistiky bežného správania, sledujú odchýlky, model sa vyvíja.

Výkonnosť expertných systémov je nízka, pretože sledovanie auditu, transformácia a odvodzovanie sú výpočtovo náročné. Preto sa komerčne nepresadzujú.

Host vs. network based IDS

Podľa umiestnenia zdrojov informácii pre IDS zariadenie rozlišujeme host based (IDS pracujúce a zamerané na hostiteľský systém) a network based (umiestnené v počítačovej sieti, spracúvajúce sieťovú premávku).

Host based IDS: Sú vývojovo staršie, ich význam bol väčší pri centralizovanom modele výpočtovej techniky. V počiatkoch boli určené do prostredia mainframe počítačov. Vplyv zvonku bol minimálny a tak zameranie bolo na lokálne útoky. Analýzou auditných informácií pozorujú podozrivé správanie. Ako zdroj informácií môžu využívať aj aplikačné logy, ktoré nezaťažujú systém nepodstatnými informáciami, ale obsahujú všetko podstatné o bežiacej službe.

Hlavnými výhodami sú silná identifikácia užívateľa, podrobné prehľady o všetkých úkonoch, vypnutie systému pri zlyhaní auditného systému.

Naopak ako nevýhodami sa ukázalo veľké množstvo auditných informácií, u ktorých len málo má bezpečnostný význam. Spracovanie takéhoto množstva informácií sa odráža aj na zvýšenej spotrebe zdrojov. Proti týmto systémom sú používané DoS útoky zamerané na preťaženie auditného systému a odstavenie služby.

Network based IDS (NIDS): Prechodom do distribuovaného výpočtového prostredia kde každý užívateľ má osobný počítač pripojený k sieti, vznikla potreba komunikácie lokálnych IDS. Komunikácia bola založená na výmene auditných informácií alebo na oznamovaní detegovaných narušení.

S príchodom Internetu začalo dochádzať k útokom na samotnú sieť. Tieto útoky zo svojej podstaty nemôžu byť odhalené host based IDS zariadeniami. Preto vznikli IDS založené na sledovaní sieťovej premávky a hľadaní podozrivých reťazcov v obsahu paketov. Ich výhodou je pokrytie veľkého počtu staníc pri nasadení na strategickom mieste v sieti.

Najefektívnejším a veľmi populárnym prístupom je prehľadávanie obsahu paketov. Prístup k dôležitým serverom sa odohráva takmer výlučne cez sieť, preto kontrolovanie paketov ešte pred príchodom na server je efektívnou cestou monitorovania. Zo svojej podstaty sú schopné odhaliť aj útoky na sieťovej vrstve – napríklad DoS útoky, ktorým by host based IDS nezabránili.

Podobne ako u firewallov aj tu sa používa stavový prístup. IDS sa chová ako aplikačná brána, analyzuje protokol na aplikačnej vrstve. Analýza je dôkladnejšia, ale spotrebúva viac zdrojov. Príkladom môže byť systém Bro [18].

Medzi hlavné nedostatky patrí obtiažna identifikácia útočníka. Túto nedokonalosť však môžeme zmierniť vhodným nasadením systému honey pot. Šifrovanie znemožňuje analýzu obsahu paketov. Aj IDS zariadenia sú často náchylné na DoS útoky a môžu byť odstavené.

Tento typ IDS sa komerčne používa, najznámejším je zrejme *Snort*, o ktorom si povieme podrobnejšie neskôr.

Existujú aj riešenia založené na oboch princípoch – vzájomnej komunikácii sieťového sledovača a analyzátora auditných informácií hostiteľa. Príkladom môže byť systém *Prelude* [19]. Je možné že sa v budúcnosti tento prístup presadí kvôli jeho komplexnosti.

Ďalšie prístupy

Ďalšie prístupy sú založené na preskúmaní celkového stavu systému – kontrole integrity dôležitých súborov (systémové, konfiguračné, binárne súbory). Na počiatku nasadenia sa vytvorí databáza zdravého systému (*fingerprint*) a vypočíta sa kontrolný súčet (zvyčajne *MD5*) vybraných súborov. Podvrhnutie alebo pozmenenie súboru spôsobí detekciu chybného

kontrolného súčtu. Overenie integrity sa vykoná na požiadanie (Tripwire) [17], alebo sa overuje v reálnom čase.

Projekt LIDS [16] využíva prístup založený na *root kite*, teda pozmenených systémových volaniach. Tieto majú za úlohu ochrániť dôležité súbory v súborovom systéme. Navyše komplexne skrýva a chráni pred ukončením určité procesy (ani administrátor ich nemôže ukončiť), zamerané na monitorovanie útokov a teda ich odhalenie útočníkom nie je možné (napr. úprava príkazu ps -ax, priečinku /proc). Zakazuje odoberať/pridávať systémové moduly, pokiaľ už jadro prešlo bootovacím procesom – tento postup sa označuje ako „kernel sealing“ (zapečatenie jadra).

Signatúry

Základnou úlohou NIDS je analýza signatúr. Signatúry obsahujú charakteristiky známych útokov. Útoky sa vyznačujú nezvyčajnými kombináciami flagov v TCP hlavičke, určitými hodnotami polí hlavičky, špecifickými reťazcami v obsahu paketu, využívaním len určitých portov. Keďže si útoky môžu byť veľmi podobné, líšiace sa len v niektorých položkách, nastáva tu snaha generalizácie, aby signatúra pokryla všetky príbuzné útoky.

Reakcie

Pokiaľ dôjde k útoku, alebo pokusu oň, základnou úlohou IDS je reakcia naň, inak by bol celý systém bezcenný. Rozlišujeme aktívne a pasívne reakcie.

Aktívne reakcie zahrňujú niekoľko spôsobov:

- **Okamžité kroky proti útočníkovi** – táto možnosť zahrňuje zakázanie prístupu útočníkovi, úpravu pozmenených konfiguračných súborov, presmerovanie útočníka do pasce honey potu. Útočníci však vo väčšine prípadov neútočia na obeť priamo, ale využívajú „tretiu stranu“, ovládnutý počítač, ktorý stojí medzi útočníkom a novou obeťou. Preto aj z právneho hľadiska niektorých štátov môže byť zakázané podnikať kroky proti zdanlivému útočníkovi, ktorý je však skrytý za nevinným z tretej strany.
- **Získavanie podrobnejších informácií o útočníkovi** – pokiaľ IDS rozpoznalo, že určitý užívateľ neoprávnene zväčšil svoje práva a jeho identita je známa, je jednoduché detailne sledovať a logovať jeho aktivitu. Táto možnosť zahŕňa sledovanie miesta a času prístupu na systém, dĺžku sedenia, spustené príkazy, otvorené súbory – vytvára sa profil útočníka. Výhodou podrobnej analýzy útočníka je možnosť opraviť potencionálne trhliny v systéme.
- **Interakcia IDS a firewallu** – je efektívnym okamžitým riešením. IDS nariadi firewallu zakázanie podozrivých IP adries. IDS môže zrušiť práve prebiehajúcu podozrivú TCP reláciu zaslaním RST flagu. Tento príznak slúži štandardne pre oznámenie chyby v TCP relácii, ale môže byť využitý aj týmto spôsobom.

Pasívnymi reakciami rozumieme logovanie upozornení, ktoré musia byť následne analyzované administrátorom. Štandardom je generovanie „reportov“ s podrobným prehľadom alebo špecifickým zameraním.

2.2.5 Snort

Je voľne šíriteľný open source IDS softwér. Patrí do kategórií NIDS a knowledge based. Jeho výhodami sú veľká rozšírenosť, silná podpora komunity a dostatok nadväzujúcich podporných nástrojov (napr. oinkmaster, snortalog, SGUIL). Tieto vlastnosti z neho robia štandard v oblasti IDS.

Využíva zásuvné moduly tzv. preprocesory, ktoré spracúvajú prijatý sieťový tok pred samotnou analýzou. Jedným z najvýznamnejších je modul riešiaci fragmentáciu. Fragmentácia je často využívaná útočníkmi pre rozloženie signatúr do viacerých paketov pre vyhnutie sa odhaleniu.

Výstup systému je spracovaný výstupnými pluginmi, ktoré riešia formát výstupných dát a následne prechádza logovacím (logging facility) a poplachovým (alerting facility) zariadením.

Snort pracuje na základe konfiguračného súboru a množiny špecifikovaných pravidiel – hľadaných signatúr (ruleset), ktoré si podrobne popíšeme.

Konfiguračný súbor definuje premenné používané v pravidlách – typickými príkladmi je vymedzenie rozsahov IP adries privátnej a externej siete.

`var HOME_NET 10.1.1.0/24` – špecifikácia hodnoty premennej HOME_NET

Pravidlá sa nachádzajú v separátnych súboroch špecifikovaných konfiguračným súborom. Každý riadok odpovedá jednému pravidlu. Pravidlo sa skladá z dvoch základných častí. Ich význam si popíšeme na príklade konkrétneho (skráteného) pravidla z kolekcie *The Bleeding Edge of Snort* [20].

Hlavička

```
alert tcp $HOME_NET any -> 213.219.122.11/32 80
```

Nesie informácie o akcii, ktorá bude vykonaná (alert), použitý protokol, zdrojové a cieľové rozsahy IP adries (tu je použitá hodnota premennej $HOME_NET z konfiguračného súboru), rozsahy portov a smer toku informácie.

IP adresy môžu byť špecifikované výčtom alebo použitím adresy a masky. Porty môžu byť špecifikované rozsahmi *1000:1020* (1000 až 1020), *1000:* (1000 a viac), *:1000* (menej ako 1001). Používa sa negačný operátor „!“ značiaci doplnok ku množine všetkých hodnôt voľby, ktorá sa označuje symbolom „any“.

Telo

```
(msg: 'ATTACK RESPONSE Zone-H.org defacement notification'; flow: established,
to_server; content:'notify_'; nocase; classtype: trojan-activity; sid: 2001616;
pcre:'/notify_(defacer|domain|hackmode|reason)=/i'; rev:6; )
```

Obsahuje výčet volieb pravidla vo formáte: *voľba: hodnota;*. Najdôležitejšie voľby sú *content* – hľadaný súvislý reťazec, *pcre* – perl compatible regulárny výraz (podrobnejšie neskôr) popisujúci vzorku hľadaných dát, *msg* – varovná správa použitá pri logovaní. Podrobný popis volieb je obsiahnutý v snort manuále [12].

Na tomto konkrétnom prípade vidieť prínos použitia regulárnych výrazov v zmenšení nepresnosti systému. Medzi voľbami content a pcre je vzťah špecializácie, čiže pravdepodobnosť false positive poplachov je nižšia pri použití regulárneho výrazu.

2.3 Vyhľadávanie reťazcov

Vyhľadávanie reťazcov je výpočtovo najnáročnejšou úlohou v IDS systéme a výrazne ovplyvňuje celkovú efektivitu. Tejto oblasti sa venuje veľká pozornosť výskumu a bolo vyvinuté veľké množstvo algoritmov a optimalizácií s rôznymi prístupmi. Avšak ani vďaka týmto snahám softwérové riešenia založené na využití obecného procesoru nie sú stále dostatočné pre doteraz stúpajúce dátové toky. Významným prínosom pre toto riešenie je využívanie obecných procesorov so stále sa zvyšujúcim počtom jadier na jednom čipe. Tento súčasný hlavný trend vo vývoji obecných procesorov prináša určité možnosti paralelizácie. V oblasti nárastu výkonu obecného procesoru sa však v najbližšej budúcnosti neočakávajú prevratné zmeny.

Radikálne riešenie tohoto problému sa naskytá presunutím realizácie tejto najnáročnejšej úlohy do hardwéru, najmä vďaka veľkým možnostiam paralelizácie. Výsledné riešenie môže existovať ako aplikačne špecifický integrovaný obvod. Technológia FPGA tu nachádza svoje uplatnenie najmä v oblasti výskumu a prináša významné úspechy. Jej výhodou je flexibilita.

Prostriedkom pre pokročilé vyhľadávanie reťazcov sú regulárne výrazy (RV).

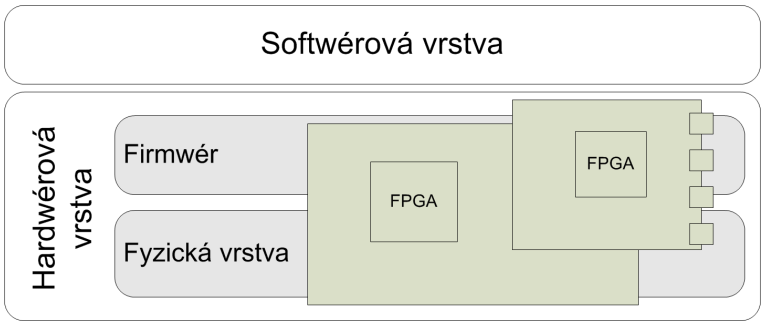
PCRE

Perl Compatible Regular Expressions [4] je knižnica implementujúca vyhľadávanie vzoriek regulárnymi výrazmi používajúca rovnakú syntax a sémantiku ako jazyk Perl. Vyjadrovacie schopnosti týchto výrazov sú značne silnejšie ako štandardné POSIX regulárne výrazy. Preto sa táto knižnica ujala v moderných programovacích jazykoch napr. Perl, C#, PHP.

2.4 Architektúra sondy Traffic Scanner

Sonda *Traffic Scanner* [5] vyvíjaná v rámci projektu *Liberouter* [6] slúži ako **hardwérový akcelerátor IDS programu Snort**. Funguje na princípe vyhľadávania jednoduchých reťazcov určených snort pravidlami, avšak nedisponuje podporou vyhľadávania regulárnych výrazov.

Architektúra sondy zobrazená na obrázku 2.1 pozostáva z niekoľkých vrstiev, ktoré budú popísané podrobnejšie:



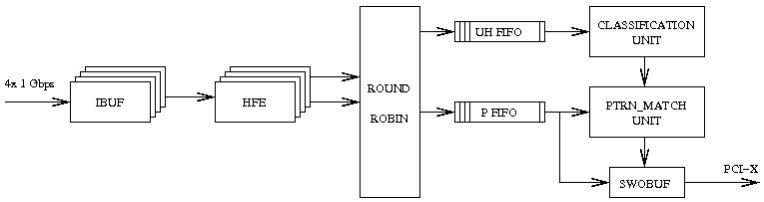
Obrázok 2.1: Vrstvy architektúry sondy Traffic Scanner

Softwérová vrstva: Táto vrstva zabezpečuje prepojenie nižších vrstiev s operačným systémom, konfiguračné a programové vybavenie pre prácu so sondou. Softwérová vrstva je podrobne popísaná v samostatnej sekcii 4.1.

Fyzická vrstva: Sonda Traffic Scanner je postavená na hardwérovej platforme *COMBO6X* – PCI-X karte využívajúcej technológiu *FPGA* (Field Programmable Gate Array). Je osadená programovateľným hradlovým poľom *Xilinx Virtex-II Pro XC2VP50*. Platforma je univerzálna, v rámci projektu Liberouter zameraná na prostredie počítačových sietí. Spojenie so sieťou zabezpečuje prídavná karta so sieťovým rozhraním a prídavným hradlovým poľom. Traffic Scanner nesie prídavnú kartu *COMBO-4SFPRO* disponujúcu štyrmi optickými gigabitovými rozhraniami.

Firmwér: Táto vrstva zabezpečuje samotnú funkcionalitu celej sondy. Obsahuje výkonné jadro systému spracúvajúce sieťovú premávku. Každý paket je preverovaný oproti množine signatúr získaných transformáciou špecifikovaných snort pravidiel. Proces transformácie nie je triviálny a bude popísaný podrobnejšie neskôr.

Systém sa skladá z menších komponent, ktoré sú implementované v jazykoch VHDL a Handel-C.



Obrázok 2.2: Firmwér sondy Traffic Scanner

Cesta paketu systémom

Paket je prijatý sieťovým rozhraním a po overení kontrolného súčtu CRC a veľkosti paketu je uskladnený vo vstupnej vyrovnávacej pamäti (IBUF). Následne je jeho hlavička rozparsovaná jednotkou HFE (Header Field Extractor - generický 16 bitový RISC procesor), dôležité informácie ako IP adresy, porty a TCP príznaky (flagy) sú uložené do štruktúry unifikovanej hlavičky. Tu končí vstupná vetva systému, ktorá sa nachádza v štyroch inštanciách (jedna pre každé rozhranie). Následne sa cesta paketu rozdelí na unifikovanú hlavičku a samotný obsah (payload) a zaradí sa do príslušných front (UH FIFO – Unified Header FIFO, P FIFO – Packet Payload FIFO).

Samotné výkonné jadro systému je obsiahnuté v klasifikačnej jednotke (CLASSIFICATION UNIT) a komponente vyhľadávajúcej reťazce v obsahu paketu (PTRN_MATCH UNIT – pattern match unit). Klasifikačná jednotka na základe informácií z unifikovanej hlavičky (zdrojové,cieľové IP adresy a porty, smer toku) rozhoduje o aplikácii len určitých pravidiel a teda prispieva k šetreniu cenného výpočtového výkonu.

Pokiaľ je paket jadrom systému označený ako splňujúci špecifikované pravidlá je predaný do výstupnej vyrovnávacej pamäte (SWOBUF) a následne transportovaný do operačného systému ovládačom cez zbernicu PCI-X.

Princíp vyhľadávania reťazcov v jednotke PTRN_MATCH

Architektúra komponenty PTRN_MATCH je založená na *nedeterministických konečných automatoch*. Popri dobrých výsledkoch je jej podstatnou vlastnosťou **možnosť rozšírenia o vyhľadávanie regulárnych výrazov** (RV).

Pre lepšie porozumenie tejto problematike v tejto sekcii budú podané základné definície a pojmy teórie konečných automatov a regulárnych výrazov.

`Definícia regulárneho výrazu:`
Nech Σ je abeceda.

- $\oslash$ je RV značiaci prázdnu množinu (prázdny jazyk)
- ε je RV jazyk $\{\varepsilon\}$
- a, kde a $\in \Sigma$, je RV značiaci jazyk {a}

Nech r a s sú RV značiace jazyky L_r a L_s, potom:

- (r.s) je RV značiaci jazyk $L = L_r L_s$ – operácia konkatenácia
- (r + s) je RV značiaci jazyk L = $L_r \cup L_s$ – operácia zjednotenie
- (r*) je RV značiaci jazyk $L = L_r{}^*$ – operácia iterácia

`Definícia konečného automatu (KA):`
KA je pätica:
M = (Q, Σ, R, s, F), kde

- Q je konečná množina stavov
- Σ je vstupná abeceda
- R je konečná množina pravidiel tvaru: pa $\rightarrow$ q, kde p, q $\in$ Q, a $\in \Sigma \cup \{\varepsilon\}$
- s $\in$ Q je počiatočný stav
- F $\subset$ Q je množina koncových stavov

`Tvrdenia:`

- Nech L je jazyk. L je regulárny jazyk (RJ), pokiaľ existuje regulárny výraz r, ktorý tento jazyk značí.
- Pre každý RV r existuje KA M, pre ktorý platí: L(r) = L(M).
- Pre každý KA M existuje RV r, pre ktorý platí L(M) = L(r).
- Nech M = (Q, Σ, R, s, F) je KA. Konfigurácia KA M je reťazec $\chi \in Q\Sigma^*$
- Deterministický KA (DKA): KA, ktorý z každej konfigurácie môže prejsť maximálne do jednej ďalšej.

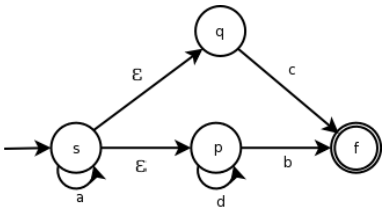
Regulárne výrazy a konečné automaty majú rovnakú vyjadrovaciu schopnosť a sú základnými modelmi pre popis regulárnych jazykov.

Implementácia nedeterministického automatu v FPGA sondy Traffic Scanner využíva ukladanie stavov v registroch a následný výber stavu pomocou kombinačnej logiky. Epsilon prechody je možné realizovať prepojením stavových registrov. 8 bitový vstup prechádza dekodérom 1 z 256. [21, 22]

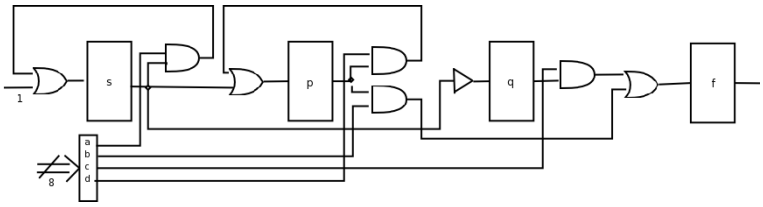
Pre ilustráciu realizácie konečného automatu v jednotke PTRN_MATCH definujeme nasledovný KA:

- Q = {s, p, q, f}
- Σ = {a, b, c, d}
- R = {sa → s, s → p, pd → p, pb → f, s → q, qc → f}
- počiatočný stav je s
- F = {f}

Obrázok 2.3 zobrazuje grafickú podobu KA a obrázok 2.4 ilustruje hardwérovú realizáciu tohoto automatu.



Obrázok 2.3: Grafické schéma nedeterministického konečného automatu



Obrázok 2.4: Hardwarová realizácia KA v jednotke PTRN_MATCH

Kapitola 3

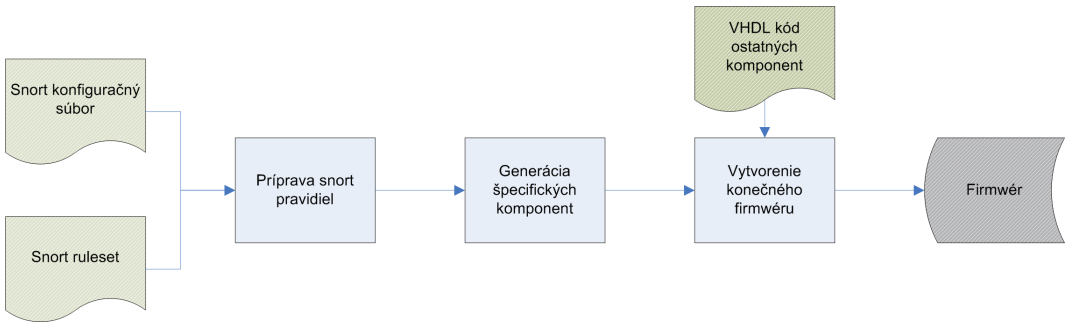
Akcelerácia vyhľadávania regulárnych výrazov

Prostriedkom pre pokročilý popis reťazcov sú regulárne výrazy. V IDS zariadeniach ich používanie reflektuje trendy generalizácie a zároveň špecializácie. V dnešnej dobe vyhľadávanie reťazcov podľa RV je v pozícii zaužívaného štandardu. Preto je nevyhnutné pri hardwérovej akcelerácii IDS zariadení implementovať túto vlastnosť.

IDS Snort využíva ku komplexnejšiemu popisu signatúr PCRE [4]. Pre poskytnutie plnohodnotnej akcelerácie programu Snort je významnou potrebou implementácia vyhľadávania reťazcov na základe PCRE. Implementácia tejto vlastnosti taktiež prispeje k zvýšeniu parametrov tohoto riešenia, najmä presnosti a kompletnosti. Naopak môže mať neblahý vplyv v podobe zvýšenia pamäťových nárokov a zníženia priepustnosti.

Akcelerácia vyhľadávania PCRE v sonde Traffic Scanner spočíva vo vytvorení špecifického firmwéru. Tento proces je znázornený na obrázku 3.1. Proces môže byť rozdelený do troch logicky oddelených častí – *príprava snort pravidiel*, *generácia špecifických komponent* a nakoniec *vytvorenie výsledného fimrwéru*.

Vstupmi procesu sú súbory so snort pravidlami (Snort ruleset), snort konfiguračný súbor a kód štandardných komponent. Výstupom procesu je firmwér pracujúci podľa špecifikovaných pravidiel.



Obrázok 3.1: Základné fázy vytvárania firmwéru Traffic Scanner

3.1 Príprava snort pravidiel

Pre prácu s dátami obsiahnutými v snort pravidlách, musia byť tieto dáta spracované a transformované do vhodných dátových štruktúr. Proces prípravy je zobrazený na obrázku 3.2.

Príprava snort pravidiel pozostáva z dvoch hlavných krokov:

Nahradenie premenných: Konfiguračný súbor obsahuje množstvo premenných, zväčša špecifikujúcich rozsahy IP adries a rozsahy portov často používaných sieťových „zón“ a služieb napr. HOME_NET (LAN), EXTERNAL_NET (Internet), HTTP_PORTS(porty služieb pracujúcich s http protokolom).

Pre nasledovné spracovanie je nutné výskyt týchto premenných v súboroch s pravidlami nahradiť hodnotami. Pre tento účel som implementoval jednoduchý bash skript s názvom *replace_var*.

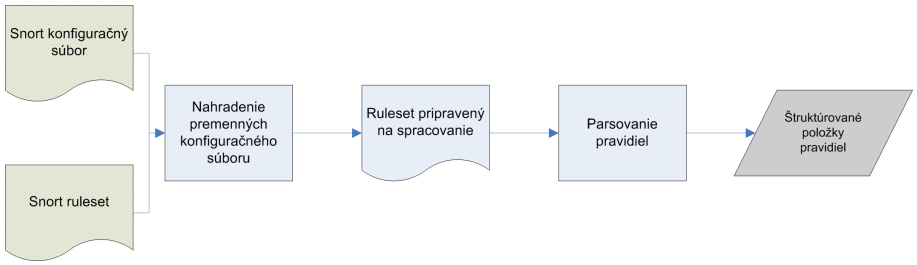
Parsovanie pravidiel: V tomto kroku dochádza k spracovaniu pravidiel a k vytvoreniu dátových štruktúr s nevyhnutnými údajmi pre generáciu špecifických komponent firmwéru – CLASSIFICATION UNIT a PTRN_MATCH UNIT.

Z hlavičky sú extrahované rozsahy IP adries, rozsahy portov, použitý protokol a smer toku sieťovej premávky. Tieto položky využívajú viac možností zápisu.

Z tela pravidla sú spracované potrebné voľby – *content*, *uricontent* a *pcre*.

Pre tento účel som implementoval potrebné dátové štruktúry a sadu funkcií v jazyku C++ s názvom *rules*.

Výsledkom tejto časti transformačného procesu sú štruktúrované dáta, ktoré sú nutným zdrojom nasledujúcej časti.

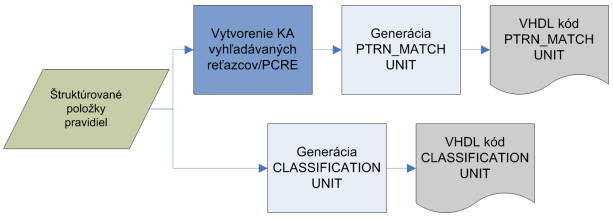


Obrázok 3.2: Proces prípravy snort pravidiel

3.2 Generácia špecifických komponent

Vstupom tejto časti transformačného procesu sú dátové štruktúry pravidiel. Výstupom je VHDL kód implementujúci unikátne komponenty PTRN_MATCH UNIT a CLASSIFICATION UNIT. Komponenta CLASSIFICATION UNIT je vygenerovaná na základe dát z hlavičiek pravidiel. Pretože proces generácie kódu tejto komponenty nevznikol v rámci tejto bakalárskej práce, v tejto sekcii sa zameriam podrobne na *proces generovania KA z PCRE* snort pravidiel a na následný prevod do kódu VHDL komponenty PTRN_MATCH.

Jednou z hlavných úloh tejto bakalárskej práce bolo vytvorenie sémantického a syntaktického analyzátoru (parseru) a pomocných knižníc transformujúcich PCRE obsiahnuté v snort pravidlách na konečný automat a následne do podoby VHDL kódu použitého v jednotke PTRN_MATCH.



Obrázok 3.3: Proces generácie špecifických komponent

3.2.1 Použité programové vybavenie

Flex

Je open source program pre generovanie lexikálnych analyzátorov. Lexikálny analyzátor je špecifikovaný regulárnymi výrazmi a akciami, ktoré sa majú vykonať pri ich rozpoznaní popísaných v jazyku C. Výsledok programu je zdrojový kód lexikálneho analyzátoru v jazyku C. Rozhranie tvorí funkcia yylex() spracúvajúca vstup. Lexikálny analyzátor rozdeľuje vstup na lexémy – logicky oddelené lexikálne jednotky, ktoré sú reprezentované „tokenmi“ ako základnými syntaktickými jednotkami.

Vygenerovaný lexikálny analyzátor využíva na rozpoznanie lexém vygenerovaný konečný automat a preto bol flex zvažovaný aj ako koncové riešenie problému generovania konečných automatov z PCRE. Táto alternatíva však narazila na nedostatky v rozdielnych syntaktických a sémantických konštrukciách regulárnych výrazov popisujúcich lexémy programu flex a PCRE (obsahujú bohatšie výrazové prostriedky). Pretože hardwérová akcelerácia nemusí riešiť všetky detaily vyhľadávania PCRE, výsledný automat môže byť zjednodušený v porovnaní s automatom vygenerovaným programom flex. V neprospech tejto alternatívy zavážila aj už implementovaná a používaná sada funkcií pre optimalizácie konečných automatov používajúca odlišné dátové štruktúry vhodné pre transformáciu do kódu VHDL.

Bison

Je open source generátor syntaktických analyzátorov (parserov). Je príbuzný programu flex najmä formátom spracovávaného súboru a často sa táto dvojica používa spolu – syntaktický analyzátor vygenerovaný programom bison využíva lexikálny analyzátor vygenerovaný programom flex. Výsledkom spracovania je zdrojový kód (samozrejme v jazyku C) LALR(1) prekladača. Prekladač je popísaný bezkontextovou gramatikou vo formáte jednotlivých gramatických pravidiel a akcií. Akcie obsahujú kód, ktorý bude vykonaný pokiaľ prekladač rozozná v spracovávaných dátach špecifikovanú gramatickú konštrukciu. Rozhranie tvoria funkcie yyparse() a yyerror().

3.2.2 Transformácia PCRE na KA

Pre vytvorenie syntaktického analyzátoru spracúvajúcemu PCRE bolo nutné špecifikovať gramatiku PCRE.

Chcel by som zdôrazniť, že snort pravidlá nevyužívajú všetky prostriedky špecifikácie reťazcov, ktoré ponúka PCRE. Taktiež hardwérová akcelerácia nevyžaduje detailné dodržovanie pokročilých vlastností PCRE. Medzi ne možno zaradiť niektoré modifikátory. Ako príklad uvádzam modifikátory /s a /m jemne upravujúce význam špeciálnych znakov $ a ^. Ich dodržovanie by vyústilo v náraste pamäťových nárokov a možnom znížení výkonu a priepustnosti výsledného riešenia. Dôsledkom tohoto kroku bude síce zvýšený počet falošných poplachov[1], ale túto situáciu nemusíme považovať za nedostatok pri modele akcelerátora pracujúceho s programom Snort. Ten redundantné dáta vylúči.

Na základe hore uvedených faktov som definoval bezkontextovú gramatiku popisujúcu potrebnú syntax PCRE použitú v implementovanom syntaktickom analyzátore. Gramatika je uvedená v prílohe A.

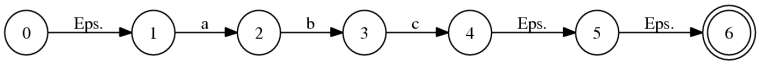
Sémantika konštrukcií PCRE

V tejto sekcii popíšem jednotlivé syntaktické konštrukcie PCRE a ich sémantiku. Ku každej transformovateľnej konštrukcii uvediem na príklade príslušný vizualizovaný konečný automat[2] vygenerovaný transformáciou regulárneho výrazu implementovaným parserom. Výsledný automat reprezentujúci komplexný PCRE výraz je zložený kombináciou týchto elementárnych konštrukcií.

- **Perl Compatible regulárny výraz:** je uzavretý pomocou znakov „/“. Mimo výrazu je možno použiť *modifikátory* upravujúce význam niektorých znakov. Pred výrazom je to modifikátor m. Za výrazom môžu stáť modifikátory i, m, s, x.

  Sémantický analyzátor využíva modifikátory i a x. Modifikátor i spôsobuje necitlivosť na veľkosť písmen (case insensitive). Modifikátor x povoľuje použitie oddeľovačov pre prehľadnosť regulárneho výrazu, ktoré nebudú zahrnuté do výsledného KA.

- **Jednoduchá konkatenácia:** značí zreťazenie po sebe idúcich znakov. Reprezentácia výrazu /abc/ je na obrázku 3.4



Obrázok 3.4: Reprezentácia výrazu /abc/

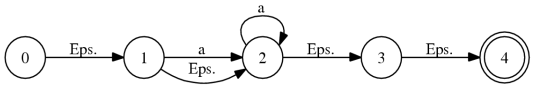
- **Zoskupenie ():** konštrukcia zoskupuje viacero elementov do skupiny. Špeciálne znaky a konštrukcie potom pracujú s celou skupinou.

- **Kvantita 0 až ∞ – špeciálny znak * :** znak, alebo skupina pred špeciálnym znakom * sa nemusí vyskytovať vôbec a môže sa opakovať nekonečno-krát. Reprezentácia výrazu /a*/ je na obrázku 3.5, výrazu /(abc)*/ na obrázku 3.6.

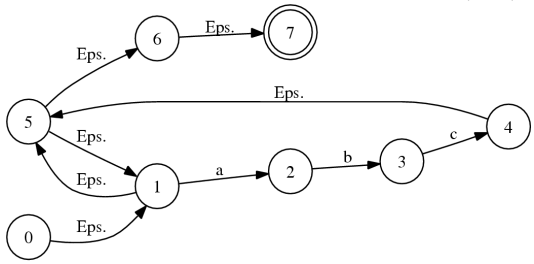
- **Kvantita 1 až ∞ – špeciálny znak + :** znak, alebo skupina pred špeciálnym znakom + sa musí vyskytovať aspoň raz. Reprezentácia výrazu /a+/ je na obrázku 3.7, výrazu /(abc)+/ na obrázku 3.8.

[1] Nejedná sa priamo o poplachy ako reakcie IDS, ale o pakety označené akcelerátorom.

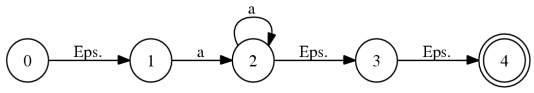
[2] Pre vizualizáciu KA vznikla sada funkcií využívajúcich dátové štruktúry vygenerovaného KA. Využíva sa program *dot* schopný generovať Post Script dokument.



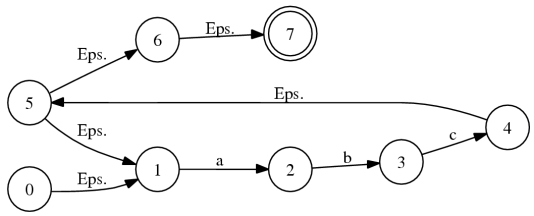
Obrázok 3.5: Reprezentácia výrazu /a*/



Obrázok 3.6: Reprezentácia výrazu /(abc)*/

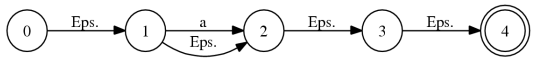


Obrázok 3.7: Reprezentácia výrazu /a+/

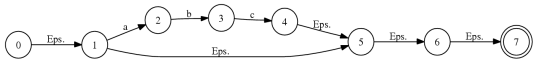


Obrázok 3.8: Reprezentácia výrazu /(abc)+/

- **Kvantita 0 alebo 1 – špeciálny znak ? :** znak, alebo skupina pred špeciálnym znakom ? sa môže vyskytovať najviac raz. Reprezentácia výrazu /a?/ je na obrázku 3.9, výrazu /(abc)?/ na obrázku 3.10.

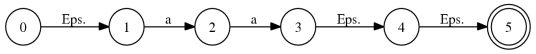


Obrázok 3.9: Reprezentácia výrazu /a?/



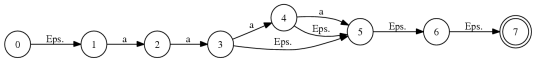
Obrázok 3.10: Reprezentácia výrazu /(abc)?/

- **Kvantita n – {n}:** znak, alebo skupina sa musí vyskytovať presne n-krát. Reprezentácia výrazu /a{2}/ je na obrázku 3.11.
- **Kvantita n až m – {n,m}:** znak, alebo skupina sa musí vyskytovať najmenej n-krát,



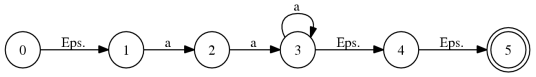
Obrázok 3.11: Reprezentácia výrazu /a{2}/

maximálne však m-krát. Reprezentácia výrazu /a{2,4}/ je na obrázku 3.12.



Obrázok 3.12: Reprezentácia výrazu /a{2,4}/

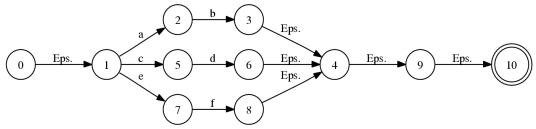
- **Kvantita n a viac – {n,}:** znak, alebo skupina sa musí vyskytovať aspoň n-krát. Reprezentácia výrazu /a{2,}/ je na obrázku 3.13.



Obrázok 3.13: Reprezentácia výrazu /a{2,}/

Konštrukcie vyjadrujúce kvantitu – *, +, {n}, {n,m}, {n,} sa môžu vyskytovať i vo variante s nasledujúcim znakom „?“. Toto použitie značí „nie chamtivý“ (not greedy) spôsob vyhľadávania. Toto chovanie hľadania prijíma čo najmenší počet opakovaní tak aby zbytok RV vyhovel. Naopak chamtivé vyhľadávanie prijíma čo najväčší možný počet opakovaní. V hardwérovej realizácii chamtivého a nie chamtivého vyhľadávania nie sú rozdiely. Aktivovanie štartovacieho stavu s každým novým prijatým znakom v konečných automatoch zaručuje súčasné vyhľadávanie obomi spôsobmi.

- **Alternácia – špeciálny znak | :** udáva možnosť výberu z viac možností – a|b – buď a alebo b. Alternovať možno ľubovolné množstvo znakov (reťazcov medzi alternáciami) alebo skupín. Reprezentácia výrazu /ab|cd|ef/ je na obrázku 3.14.



Obrázok 3.14: Reprezentácia výrazu /ab|cd|ef/

- **Znakové skupiny:** umožňujú výber práve jedného zo skupiny znakov. Využívajú viac foriem zápisu – vymenovaním konkrétnych znakov, intervalom s použitím znaku „-“, štandardnými POSIX znakovými skupinami ([:digit:], [:space:]) známymi z jazyka C definovanými funkciami ako isdigit() alebo isspace(). Tieto tri formy zápisu sa používajú v konštrukcii [ ]. V poslednom prípade PCRE prináša možnosť zápisu escape sekvenciami. Nasledujúce štyri zápisy vyjadrujú rovnakú skupinu znakov vymenovanými spôsobmi zápisu.

```
/[0123456789]/ /[0-9]/ /[[:digit:]]/ /\d/
```

Znakové skupiny dovoľujú využívať i negačný operátor „^“ v zmysle prípustné je všetko okrem definovanej skupiny. Escape sekvencie negáciu vyjadrujú použitím veľkého písmena.

```
/[^0123456789]/ /[^0-9]/ /[[:^digit:]]/ /\D/
```

Skupina so zápisom využívajúcim konštrukciu [ ] môže obsahovať ľubovolné množstvo kombinácií rôznych zápisov.

```
/[_abc[:digit:]A-Z]/
```

- **Špeciálny znak . :** značí ľubovolný znak.
- **Špeciálny znak $:** značí začiatok reťazca alebo riadku.
- **Špeciálny znak ^ :** značí koniec reťazca alebo riadku.

  Význam špeciálnych znakov ., $ a ^ detailne ovplyvňujú modifikátory /s a /m.
- **Escape sekvencie :** okrem už spomenutých významov slúžia pre vyjadrenie ASCII znaku zhodného so špeciálnymi znakmi. Ľubovolný ASCII znak je taktiež možné vyjadriť jeho ASCII hodnotou v osmičkovej alebo šestnástkovej číselnej sústave v tvare \0ccc (pre oktálovú, c je okta-číslica), \xcc (pre hexadecimálnu, c je hexa-číslica).

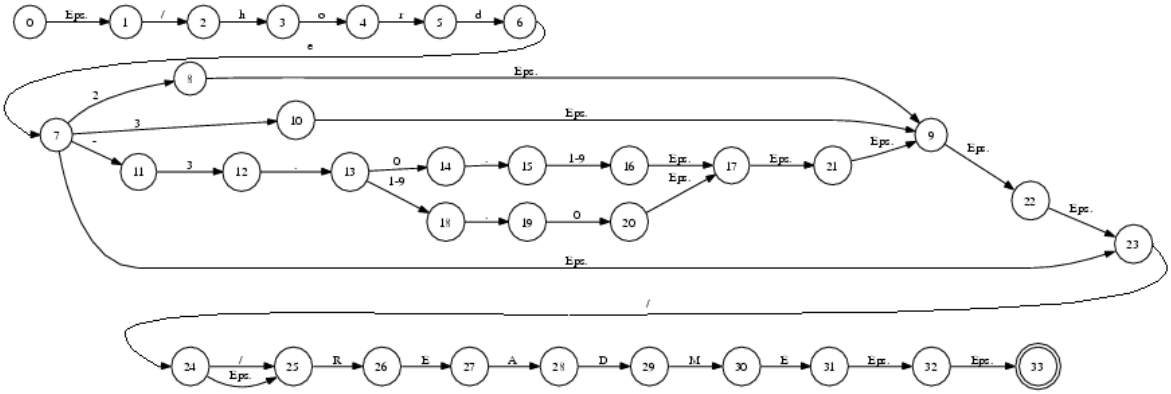
Kombináciou jednotlivých konštrukcií je možno spracovať komplexný Perl Compatible regulárny výraz použitý v konkrétnom snort pravidle. Príkladom je nasledujúce pravidlo a výsledná reprezentácia je na obrázku 3.15.

alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS (msg:'BLEEDING-EDGE WEB Horde README access – Probe'; flow:to_server,established; uricontent:'/horde'; nocase; **pcre:'/\/horde((2|3|-3\.(0\.[1-9]|1\.0)))?\/{1,2}README/';** classtype:web-application-activity; reference:cve,CVE-2006-1491; reference:url,csirt.terradon.com/postarchive.php?month=4&year=2006#article28; sid:2002897; rev:2;)

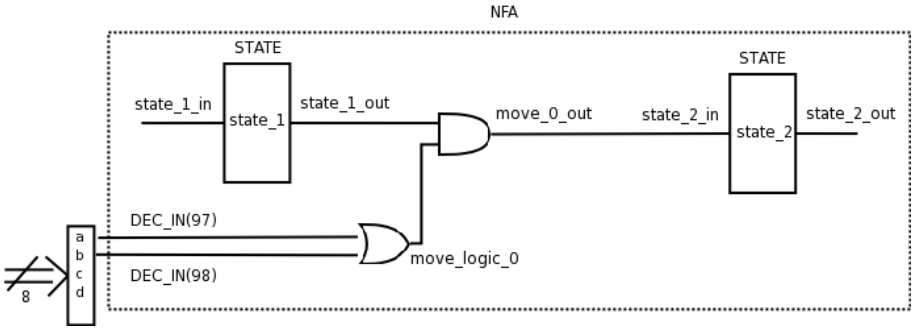
3.2.3 Transformácia KA na kód VHDL

Na obrázku 3.16 je znázornená realizácia konečného automatu v kóde VHDL. Výsledný kód popisuje architektúru komponenty *NFA* realizujúcej funkciu konečného automatu využívajúcu zdieľaný dekodér [10]. Architektúru komponenty tvoria menšie komponenty *END_STATE* a *STATE* reprezentujúce stavy KA. Prechody medzi stavmi sú realizované jednoduchým prepojením výstupu zdrojového stavu (state_0_out) a upraveného vstupu z dekodéra (move_logic_0) logickým členom *and* (výsledkom je move_0_out) pripojeným na vstup cieľového stavu (state_2_in). Možnosť uskutočnenia prechodu s viacerými znakmi je realizovaná logickým členom *or* združujúcim vstup týchto znakov (move_logic_0).

Súčasťou tejto bakalárskej práce je implementácia funkcií pre transformáciu KA do kódu VHDL. Funkcie generujú kód komponenty *NFA* uvedeným spôsobom z dátových štruktúr konečného automatu vygenerovaného parserom PCRE.



Obrázok 3.15: Reprezentácia výrazu /\/horde((2|3|-3\.(0\.[1-9]|1\.0)))?\/{1,2}README/

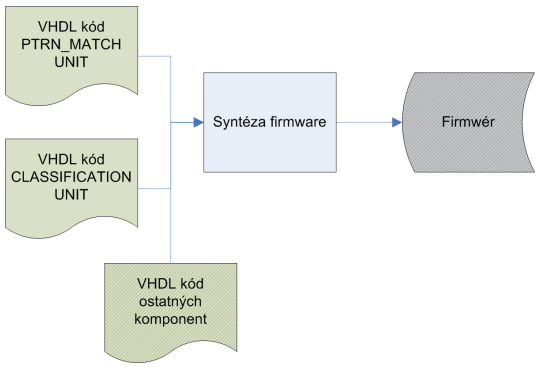


Obrázok 3.16: VHDL realizácia prechodu [ab]

3.3 Vytvorenie výsledného firmwéru

Vytvorenie funkčného firmwéru je posledným krokom transformácie snort pravidiel. Proces popisuje obrázok 3.17. Kód unikátnych vygenerovaných komponent PTRN_MATCH a CLASSIFICATION UNIT spolu s kódom ostatných štandardných komponent je skompilovaný a syntetizovaný nástrojmi Precision, XST alebo Leonardo. Tento proces je časovo náročný.

Pre dokončenie realizácie hardwérového akcelerátoru vyhľadávania PCRE Traffic Scanner je potreba posledného kroku. Týmto krokom je úprava dekodéra 1 z 256 jednotky PTRN_MATCH, ktorý privádza vstup konečného automatu (jednotlivé znaky obsahu prehľadávaného paketu). Dekodér musí byť doplnený o signály reprezentujúce znakové skupiny regulárnych výrazov popísané v predchádzajúcej sekcii.



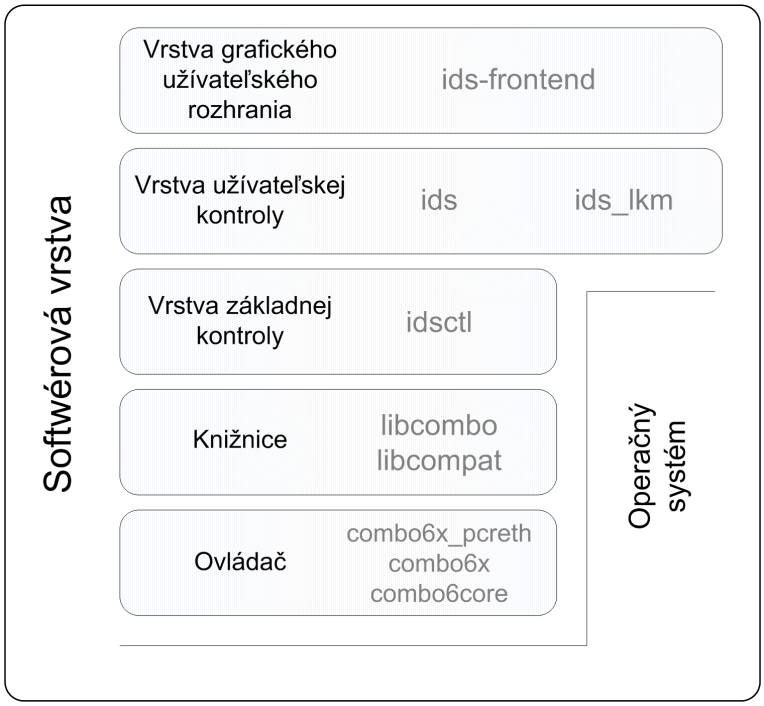
Obrázok 3.17: Proces vytvorenia výsledného firmwéru

Kapitola 4

Programové vybavenie

V tejto kapitole bude popísaná softwérová vrstva sondy Traffic Scanner a podrobnejšie každá jej zložka. Významnú časť tejto vrstvy tvorí programové vybavenie pre konfiguráciu, monitorovanie stavu a všeobecné používanie sondy Traffic Scanner.

4.1 Softwérová vrstva sondy Traffic Scanner



Obrázok 4.1: Softwérová vrstva sondy Traffic Scanner

Softwérová vrstva je zobrazená na obrázku 4.1. Používa hierarchické usporiadanie jednotlivých zložiek, kde každá vrstva využíva volania všetkých nižšie postavených vrstiev.

Použitou softwérovou platformou je operačný systém `Linux`.

Ovládač

Funkciu ovládača (driver) plnia moduly jadra. Sú základnou vrstvou zabezpečujúcou prepojenie hardwéru s operačným systémom. Koncepcia sondy Traffic Scanner je navrhnutá tak, že karta v operačnom systéme figuruje ako štandardné sieťové rozhranie (c6eth00). Tento prístup je transparentný z pohľadu akcelerácie programu Snort, ktorý pracuje nad týmto rozhraním a spracúva enormne zredukované množstvo sieťovej komunikácie vďaka hardwérovej akcelerácii.

Úlohou ovládača je transportovať pakety označené firmwérom ako obsahujúce vyhľadávané reťazce z výstupnej vyrovnávacej pamäte (SW_OBUF) do operačného systému cez zbernicu PCI-X.

Knižnice libcombo a libcompat

Knižnice libcombo a libcompat zastrešujú základné operácie s kartami rodiny COMBO. Sú implementované v jazyku C. Umožňujú načítanie daného firmwéru do čipu FPGA na karte i prídavnej karte nesúcej sieťové rozhrania, monitorovanie stavu karty, rezerváciu prístupu k zariadeniu a alokáciu firmwérových komponent. Ďalšími funkciami sú správa pamäťového priestoru firmwéru a vstupno-výstupné operácie s pamäťou a registrami karty. Pre prácu s touto knižnicou je nutné pre každý firmwér vytvoriť XML súbor popisujúci použité komponenty a ich umiestnenie v pamäťovom priestore čipu FPGA.

Tieto knižnice slúžia ako základné rozhranie pre programy pracujúce s kartami COMBO.

Vrstva základnej kontroly

Táto vrstva obsahuje program *idsctl*. Je implementovaný v jazyku C. Pracuje s registrami a pamäťou firmwérovej vrstvy Traffic Scanner. Jeho účelom je inicializácia registrov a pamäte, operácie ako reštart sondy, zapínanie a vypínanie určitých komponent (IBUF), povolovanie a zakazovanie vlastností chovania sondy a vyčítavanie stavu významných registrov. Táto vrstva slúži pre zobrazovanie štatistík významných ako ladiace informácie a okamžitého vnútorného stavu sondy.

Program pre komunikáciu využíva neinteraktívne textové rozhranie príkazového riadku. Operácia, ktorá má byť uskutočnená, je špecifikovaná argumentami príkazového riadku.

Vrstva užívateľskej kontroly

Do tejto vrstvy spadajú bash skripty *ids* a *ids_lkm*. Komunikácia je neinteraktívna prostredníctvom príkazového riadku a parametrov programov.

ids – zabezpečuje manažment dostupných firmwérov, nahrávanie zadaného firmwéru do čipu FPGA karty a poskytovanie informácií o aktuálnom firmwéry. Pre svoju činnosť overuje prítomnosť potrebných modulov jadra. Medzi ďalšie funkcie patrí konfigurácia sieťového rozhrania – zmena parametrov, jeho zapínanie a vypínanie. Ako zdroj a úložný priestor potrebných informácií využíva konfiguračný súbor *ids.conf*.

ids_lkm – stará sa o načítavanie a odstraňovanie modulov jadra potrebných pre beh sondy.

Vrstva grafického užívateľského rozhrania

Vrstva grafického užívateľského rozhrania (GUI) obaľuje vyššie popísané programové vybavenie – program idsctl, skripty ids a ids_lkm. Sprístupňuje ich funkcionalitu

v užívateľsky prístupnejšej a prehľadnejšej forme. Dôležitou schopnosťou je sprístupnenie generovania nového špecifického firmwéru vyhľadávajúceho užívateľom špecifikované signatúry.

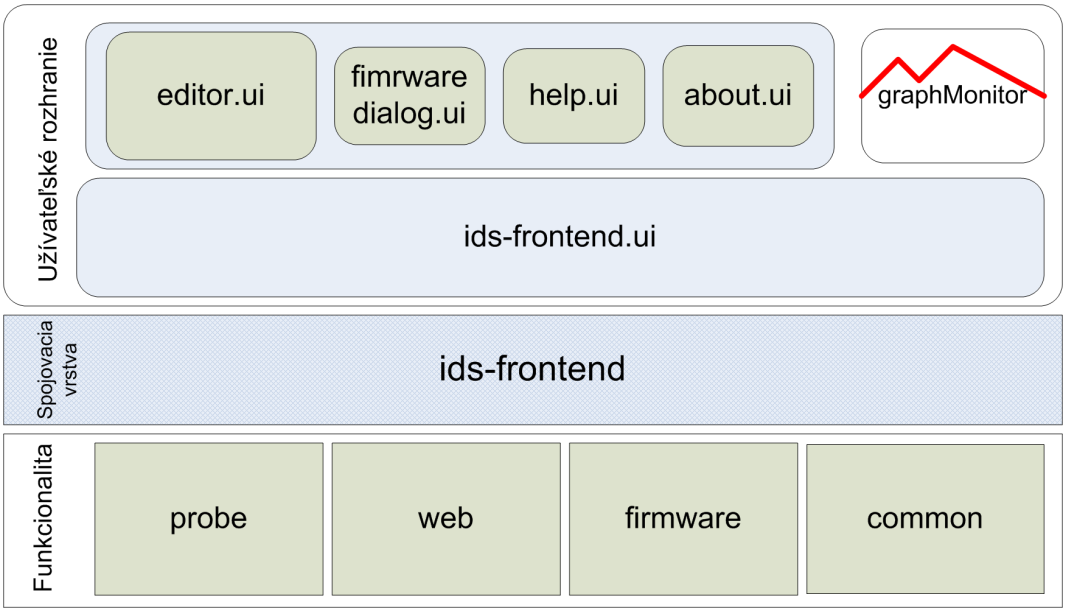
GUI nesie názov *ids-frontend* a je implementované v jazyku C++ s použitím knižníc qt4, ktoré poskytujú triedy grafického rozhrania i triedy s komplexnou funkcionalitou. Implementácia využíva objektový prístup.

Implementácia tejto vrstvy bola časovo náročnou časťou bakalárskej práce. Ukážky grafického prostredia aplikácie sú uvedené v prílohe B.

Qt4 je súbor knižníc v jazyku C++ tzv. „framework“ pre vývoj multiplatformových aplikácií využívajúcich GUI. Qt používa pre komunikáciu udalostami riadeného chovania objektov systém „signálov a slotov“. Táto koncepcia umožňuje nezávislý vývoj GUI a funkcionality aplikácií a možnosť „komponentného“ programovania. Pre zaručenie systému signálov a slotov a ďalších vlastností objektov je zdrojový kód obalený pridaným kódom tzv. „runtime“.

Vďaka svojím dobrým vlastnostiam bolo qt4 vybrané pre implementáciu GUI pre operácie so sondou Traffic Scanner.

4.1.1 Objektový model užívateľského rozhrania



Obrázok 4.2: Objektový model ids-frontend

Objektový model aplikácie ids-frontend pozostáva z troch hlavných vrstiev. Každú vrstvu si podrobnejšie popíšeme:

Vrstva užívateľského rozhrania: Táto vrstva bola implementovaná nástrojom designer-qt4, ktorý slúži pre vizuálne konštruovanie užívateľského rozhrania. Rozhranie je úplne oddelené od funkčného kódu. Je reprezentované súbormi *.ui* vo formáte XML.

Základným GUI prvkom je formulár ids-frontend.ui. Ostatné formuláre sú zobrazené na základe interakcie so základným formulárom.

Hlavný formulár zabezpečuje zobrazovanie najdôležitejších ukazateľov stavu sondy – prítomnosť systémových modulov, stav firmwéru a monitor počtu paketov označených sondou ako splňujúce špecifikované pravidlá. Ďalej umožňuje nahrať do sondy nový firmwér a významnou úlohou je aj spojenie so službou ids.liberouter.org. Formulár poskytuje informácie o hodnotách významných registrov zariadenia.

Základné užívateľské rozhranie taktiež obsahuje aj hlavné menu programu umožňujúce zobraziť nápovedu a korektne ukončiť aplikáciu.

graphMonitor je špecifický prvok užívateľského rozhrania použitý v hlavnom formulári. Vykresľuje prehľadný graf počtu exportovaných paketov v čase. Realizuje funkciu grafickej komponenty (widget).

Funkcie ostatných formulárov zabezpečujú zobrazenie jednoduchého editoru za účelom špecifikácie vlastných snort pravidiel, zobrazenie html dokumentácie k sonde Traffic Scanner, overenie prítomnosti potrebných súborov pri nahrávaní firmwéru a zobrazenie informácií o programe a autorovi.

Spojovacia vrstva: Je implementovaná triedou ids-frontend. Funkciou tejto vrstvy je vytvoriť potrebné objekty a spojiť reakcie užívateľského rozhrania s funkcionalitou programu. Taktiež zaručuje zobrazovanie ostatných formulárov pri interakcii s užívateľom.

Pri ukončení programu sa stará o uvoľnenie pamäte.

Funkcionalita: Funkcionálna vrstva obsahuje triedy implementujúce konkrétne nevyhnutné operácie. Hlavné úlohy týchto tried sú pravidelné vyčítavanie hodnôt registrov z firmwéru pomocou knižničnej vrstvy libcombo a libcompat, nahranie zvoleného firmwéru do čipu FPGA, komunikácia protokolom HTTP s webovým rozhraním Traffic Scanner ids.liberouter.org.

Pre komunikáciu protokolom HTTP je použitá trieda knižnice qt4 QHttp zabezpečujúca spojenie a prenos dát protokolom HTTP. Použitý je príkaz protokolu HTTP POST. Obsah príkazu je definovaný na základe užívateľského vstupu a podľa toho je zostavený výsledný paket.

Funkcionálna vrstva tiež zabezpečuje kontroly prítomnosti systémových modulov, ich načítanie, kontrolu existencie konfiguračného súboru ids.conf a parsovanie hodnôt konfiguračných premenných.

Pre splnenie niektorých účelov využíva nižšiu vrstvu softwérového modelu Traffic Scanner – skripty ids a ids_lkm. Pre ich volanie využíva funkciu system().

Program ids-frontend obsahuje priamo užívateľskú dokumentáciu a je možné ju zobraziť pomocou hlavného menu. Je použitý vizuálny formát s popisujúcim textom (screenshot).

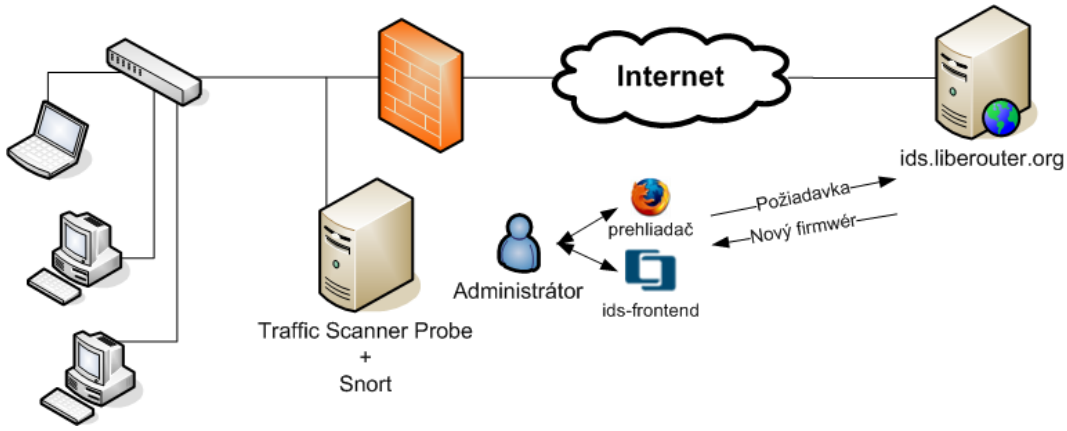
Kapitola 5

Detekcia narušenia siete s využitím Traffic Scanner

Táto kapitola bude zameraná na zariadenie Traffic Scanner z pohľadu funkčného bezpečnostného prvku sieťovej infraštruktúry. Riešená bude otázka zapojenia zariadenia, užívateľská koncepcia a vízia spolupráce s inými bezpečnostnými zariadeniami vyvíjanými projektom Liberouter.

5.1 Užívateľská koncepcia

Základnou požiadavkou užívateľov je potreba aktuálnosti vyhľadávanej množiny pravidiel a možnosť vytvorenia vlastných pravidiel. Pre tieto potreby bol zriadený server ids.liberouter.org. Prebieha tu celý proces transformácie snort pravidiel na firmwér na základe požiadavky užívateľa. Interakcia s užívateľom je riešená webovým rozhraním implementovaným v jazyku PHP. Komunikácia je založená na protokole HTTP.



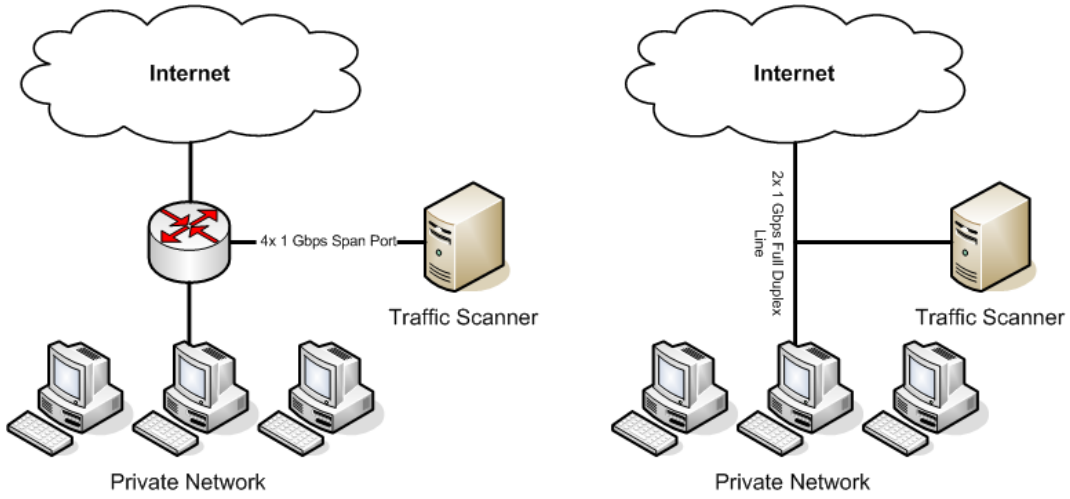
Obrázok 5.1: Užívateľská koncepcia Traffic Scanner

K rozhraniu je možno pristupovať pomocou internetového prehliadača. O výsledku a možnostiach stiahnutia výsledkov požiadavku je užívateľ informovaný elektronickou poštou. Je taktiež možné využiť program ids-frontend, ktorý zabezpečuje potrebnú HTTP komunikáciu. Poskytuje rovnaké možnosti ako webové rozhranie.

Pretože proces transformácie snort pravidiel na syntetizovaný firmwér je veľmi výpočtovo náročný, pri stúpajúcom množstve užívateľov bude musieť byť koncepcia upravená.

5.2 Spôsob zapojenia v sieti

Existujú dva možné spôsoby zapojenia. Sú zobrazené na obrázku 5.2. Štvor-portové monitorovanie využíva prijímanie mirrorovaného sieťového toku všetkými sieťovými rozhraniami. Pri spôsobe zapojenia „inline“ všetok sieťový tok prechádza sondou obojsmerne. Sonda sa chová ako „T-splitter“.



Obrázok 5.2: Sieťové zapojenie

5.3 Vízia spolupráce s inými bezpečnostnými zariadeniami

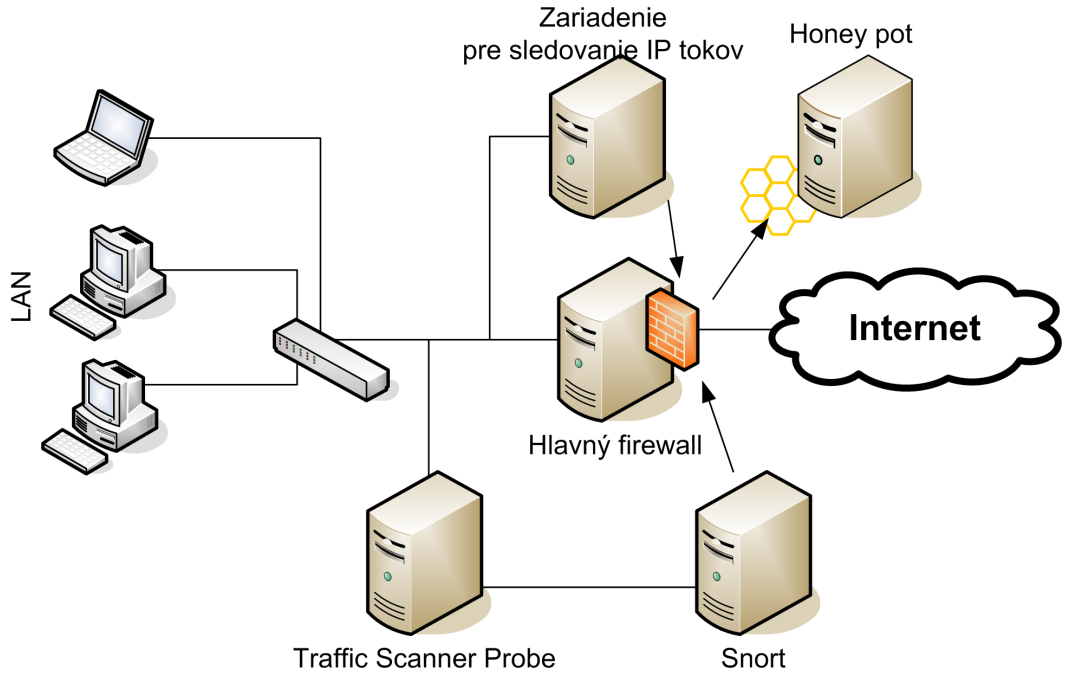
Zariadenie Traffic Scanner má určite potenciál pre použitie aj s inými bezpečnostnými zariadeniami, ktoré by pomohli zvýšiť bezpečnosť výsledného systému. Táto koncepcia je zobrazená na obrázku 5.3. Potencionálna spolupráca zariadenia Traffic Scanner sa naskytá v použití s nasledovnými zariadeniami:

- **Firewall** – perspektíva spolupráce zariadenia s riešením IDS Traffic Scanner je vysoká. IPS zložka systému Snort, by mohla vzdialene konfigurovať hlavný firewall a aktívne tak odpovedať na hrozby. Taktiež by IPS zložka pri zistení pokusu o útok mohla pomocou firewallu presmerovať útočníka do pasce honey potu alebo padded cell systému.
- **Zariadenie pre sledovanie IP tokov** – je pasívne sieťové monitorovacie zariadenie. Zbiera štatistiky o IP tokoch (flow) a exportuje ich. Potenciálna spolupráca vyplýva z perspektívy zariadenia detegovať DoS útoky, ktoré sa vyznačujú veľkým počtom IP tokov v malom časovom okamihu. Táto spolupráca by mohla zvýšiť bezporuchovosť sondy Traffic Scanner, pretože toto zariadenie môže byť cieľom práve DoS útokov. Samozrejme predpokladá použitie firewallu pre realizáciu protiopatrení.

- **Honey pot alebo Padded Cell System** – ako už bolo napísané, v spolupráci s firewallom by toto zariadenie mohlo slúžiť ako pasca pre získavanie informácií o útočníkoch.

Súčasný stav neumožňuje spoluprácu zariadení. Najväčším nedostatkom je rozdielnosť konfiguračných rozhraní pre rôzne bezpečnostné zariadenia, najmä firewally. Táto situácia by si vyžadovala vznik ďalšieho softwéru, ktorý by zjednocoval rozdielne rozhrania a vytváral unifikované rozhranie. Perspektívnym prostriedkom pre túto úlohu môže byť protokol *netconf*.

Táto úloha by bola značne náročná a nad rámec tejto bakalárskej práce.



Obrázok 5.3: Koncepcia spolupráce s inými bezpečnostnými zariadeniami

Kapitola 6

Záver

Naštudovaná bola problematika sieťovej bezpečnosti a najmä zariadení IDS. Podrobne bola rozobraná architektúra hardwérového akcelerátoru IDS programu Snort, Traffic Scanner, vyvíjaného v rámci projektu Liberouter. Ako súčasť procesu transformácie snort pravidiel na firmwér Traffic Scanner bola implementovaná sada funkcií transformujúca položky snort pravidiel do dátových štruktúr vhodných pre ďalšie spracovanie. Pre zabezpečenie vyhľadávania reťazcov na základe PCRE bola vytvorená potrebná gramatika popisujúca pcre zložky pravidiel snort a na jej základe bol implementovaný syntaktický a sémantický analyzátor, ktorý je schopný generácie konečných automatov do dátových štruktúr vhodných k transformácii do kódu VHDL. Tieto dátové štruktúry sú spracovávané implementovanou sadou funkcií na prevod konečných automatov do kódu VHDL. Ďalej bolo popísané a implementované programové vybavenie sondy, ktorého časovo a implementačne najnáročnejšiu časť tvorí aplikácia s komplexným užívateľským rozhraním. Ku všetkým implementovaným nástrojom bola v jazyku docbook vytvorená užívateľská dokumentácia vo formáte manuálových stránok systémov UNIX. Bola tiež analyzovaná možnosť spolupráce Traffic Scanner s inými bezpečnostnými zariadeniami. Nakoniec bola predvedená koncepcia detekcie narušenia siete s využitím sondy.

Oproti pôvodnému zadaniu, kde bol značný dôraz kladený na spoluprácu akcelerátora s inými bezpečnostnými zariadeniami, sa práca zamerala najmä na podporu vyhľadávania regulárnych výrazov. Hlavným dôvodom k tomuto kroku je nevyhnutnosť hardwérovej podpory vyhľadávania reťazcov na základe regulárnych výrazov. Táto podpora zo strany IDS zariadení je pevným štandardom a z hardwérovej strany potrebou pre poskytnutie akcelerácie softwérovým IDS nástrojom. Bez tohoto kroku by spolupráca s inými zariadeniami poskytovala zlé výsledky, akcelerátor by zabránil odhaleniu niektorých útokov.

Prostriedky pre analýzu v reálnom čase, ukladanie dát i pre off-line spracovanie poskytuje samotný program Snort a jeho podporné nástroje. Snort je schopný ukladať dáta do databáze MySQL i zasielať upozornenia o útokoch v reálnom čase. Podporný nástroj SGUIL poskytuje GUI prostredie pre analýzu dát v reálnom čase i pre vykonávanie off-line operácií nad databázou. Nástroj SGUIL bol nainštalovaný a vyskúšaný s programom Snort pre daný účel.

Smery pokračovania tejto bakalárskej práce sú početné. Veľký potenciál v ušetrení pamäťových nárokov poskytujú optimalizácie použitých konečných automatov alebo zmena princípu ich hardwérovej realizácie. Ďalšou potrebou riešenia Traffic Scanner je poskytnutie aktuálnosti vyhľadávaných signatúr a teda nabáda ku vzniku programového vybavenia zabezpečujúceho túto úlohu. Súbežne s predpokladaným vytváraním potrebného vybavenia pre kontrolu ostatných zariadení projektu Liberouter rastú možnosti ich reálnej spolupráce.

Príloha A

Implementovaná gramatika PCRE

Terminály: ASCII[1], /, *, +, ?, OR[2], ., $, ^, [,], (,), {, }, INT[3], CHARCLASS2VALUE[4], CHARCLASS[5], NEGATE[6], SLASHCHARCLASS[7], NEGSLASHCHARCLASS[8], SPACE[9]

Nonterminály: pcre, modif_front, modif_rear, inslash, exp, ext_unit, unit, grouping, element, class, quantity, repeating, not_greedy, interval, inclass, inclass_unit

Gramatické pravidlá:

```
pcre:           modif_front inslash modif_rear
modif_front:    /* empty */ | ASCII
modif_rear:     /* empty */ | ASCII modif_rear
inslash:        / exp /
exp:            ext_unit | ext_unit exp | ext_unit OR exp
unit:           element | grouping
ext_unit:       unit | unit repeating not_greedy | unit quantity not_greedy
grouping:       ( exp )
element:        ASCII | . | $ | ^ | SPACE | class
class:          [ inclass ] | [ NEGATE inclass ] | SLASHCHARCLASS |
                NEGSLASHCHARCLASS
quantity:       * | + | ?
repeating:      { interval }
not_greedy:     /* empty */ | ?
interval:       INT | INT , | INT , INT
inclass:        inclass_unit | inclass_unit inclass
inclass_unit:   ASCII | CHARCLASS2VALUE | CHARCLASS
```

Dôležitá je najmä **celková hierarchia** nonterminálov od základných jednotiek po kompletný regulárny výraz:

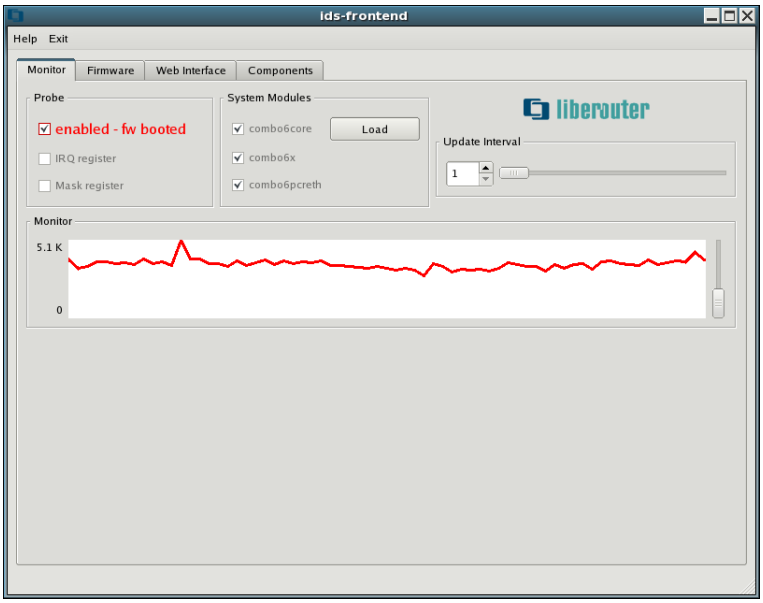
[1] ASCII znak.
[2] Znak | značiaci alternáciu.
[3] Celočíselná hodnota.
[4] Trieda znakov určená intervalom.
[5] Štandardná trieda znakov POSIX.
[6] Znak ^ negácia skupiny znakov.
[7] Trieda znakov PCRE.
[8] Negovaná trieda znakov PCRE.
[9] Medzera pre prehľadnosť – používané len s modifikátorom /x.

`element` | `grouping` → `unit` → `extended unit (ext_unit)` → `expression (exp)` → `pcre`

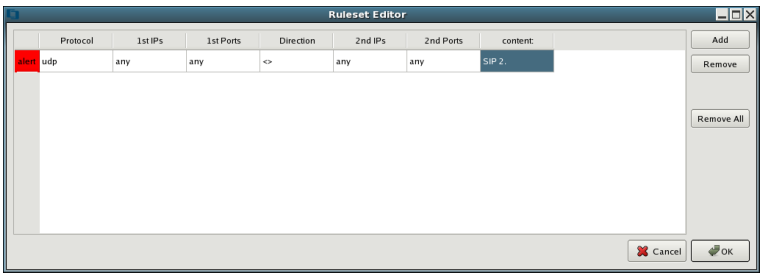
Poznámka: /* empty */ značí možnosť nonterminál zanedbať. Znak | značí alternatívu – možnosť aplikácie rozličných pravidiel.

Príloha B

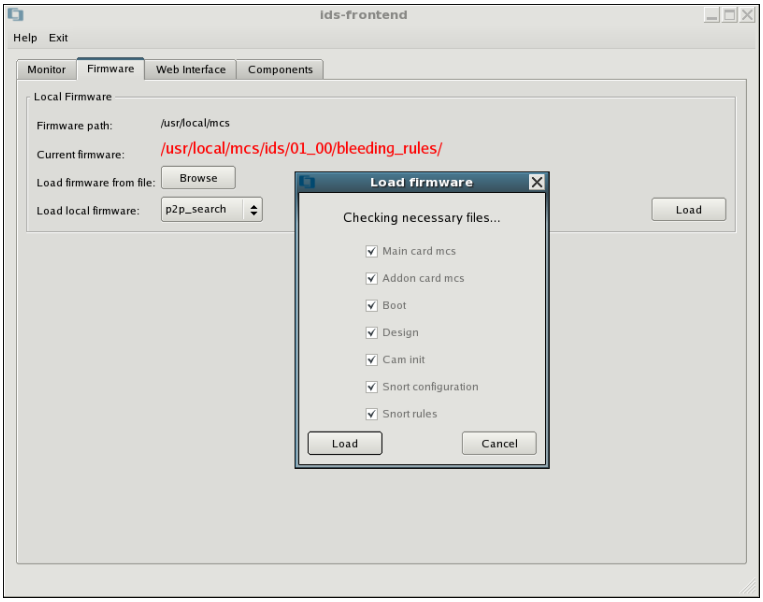
Ukážky grafického prostredia ids-frontend



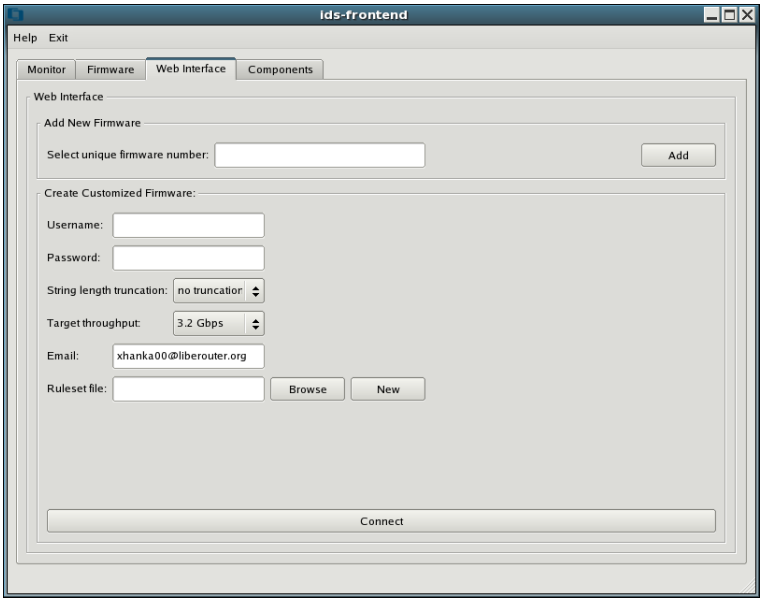
Obrázok B.1: Záložka monitor hlavného užívateľského rozhrania



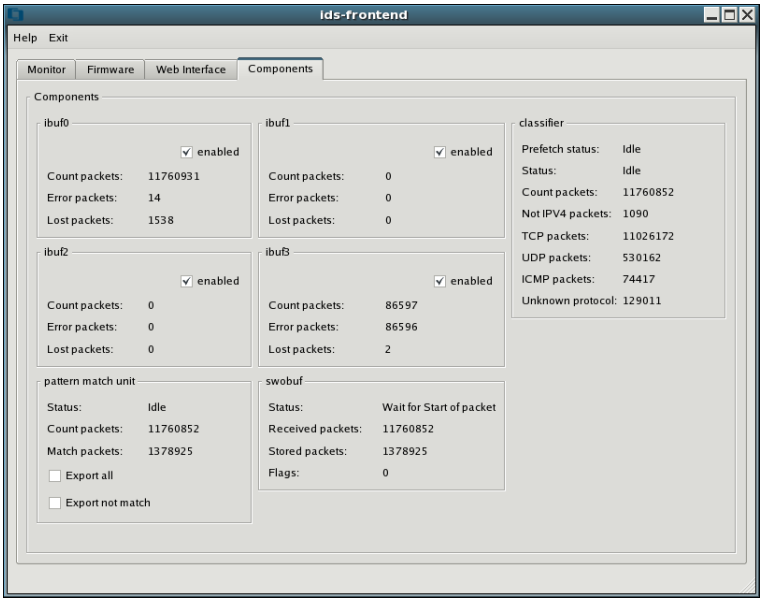
Obrázok B.2: Jednoduchý editor pravidiel snort



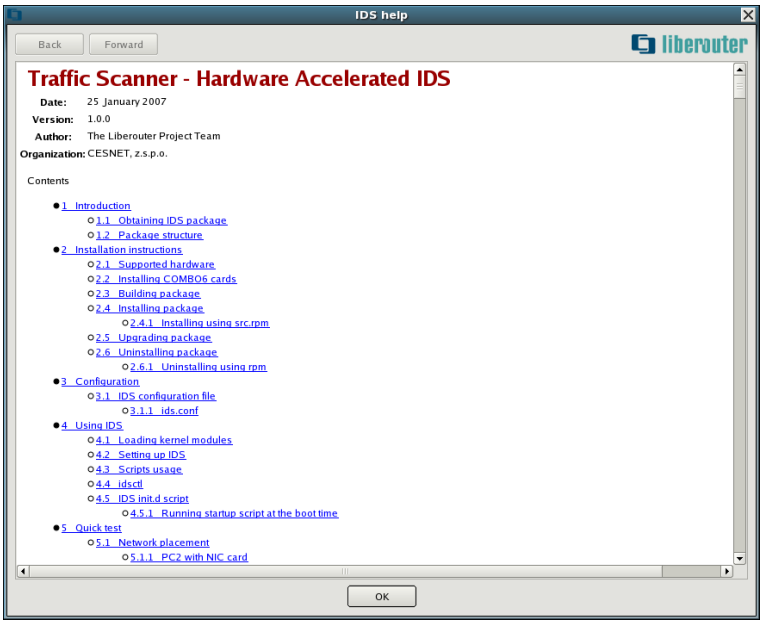
Obrázok B.3: Kontrola potrebných súborov pri nahrávaní firmwéru



Obrázok B.4: Rozhranie pre spojenie so službou ids.librouter.org



Obrázok B.5: Monitorovanie stavu registrov sondy



Obrázok B.6: HTML prehliadač dokumentácie

Literatúra

[1] Klaus Müller, Hubert Kaiβer; Intrusion Detection System, Part I. LinuxFocus Magazine, http://www.linuxfocus.org/English/Archives/lf-2003_05-0292.pdf, 2003.

[2] Klaus Müller, Jürgen Pohl; Intrusion Detection System, Part II. LinuxFocus Magazine, http://www.linuxfocus.org/English/Archives/lf-2003_07-0294.pdf, 2003.

[3] Hervé Debar; Introduction to Intrusion-Detection Systems. IBM Research, Zurich Research Laboratory.

[4] perlre – Perl regular expressions. http://perldoc.perl.org/perlre.html, 2006.

[5] IDS accelerator Traffic Scanner. http://www.liberouter.org/ids.php.

[6] Liberouter project. http://www.liberouter.org/.

[7] Fang Yu, Zhifeng Chen, Yanlei Diao, T.V. Lakshman, Randy H. Katz; Fast and Memory-Efficient Regular Expression Matching for Deep Packet Inspection. Electrical Engineering and Computer Sciences University of California at Berkeley, http://www.eecs.berkeley.edu/Pubs/TechRpts/2006/EECS-2006-76.html, 2006.

[8] Benjamin C. Brodie, Ron K. Cytron and David E. Taylor; A Scalable Architecture For High-Throughput Regular-Expression Pattern Matching. Exegy, Inc., 2006.

[9] Jakub Mahdal; Analýza dat IDS systémů. Vysoké učení technické v Brně, Fakulta Informačních technolologií, 2006.

[10] Petr Kobierský; Návrh architektury IDS systému pro technologii FPGA. Vysoké učení technické v Brně, Fakulta Informačních technolologií, 2006.

[11] Common Vulnerabilities and Exposures. http://cve.mitre.org/.

[12] Snort Manual. http://www.snort.org/docs/snort_manual/.

[13] Flex Manual. http://www.gnu.org/software/flex/manual/html_mono/flex.html.

[14] Bison Manual. http://www.gnu.org/software/bison/manual/html_mono/bison.html.

[15] Michael Sobirey; List of IDS solutions. http://www.cse.sc.edu/research/isl/mirrorSobireys.shtml.

[16] Xie Huagang; LIDS Hacking HOWTO, http://www.lids.org/lids-howto/lids-hacking-howto.html, 2000.

[17] Tripwire. http://www.tripwire.com/.

[18] Bro Intrusion Detection System. http://www.bro-ids.org/.

[19] Prelude Hybrid IDS project. http://www.prelude-ids.org/.

[20] The Bleeding Edge of Snort. http://www.bleedingsnort.com/.

[21] Ch. Clark and D. Schimmel; Efficient Reconfigurable Logic Circuits for Matching Complex Network Intrusion Detection Patterns In Field Programmable Logic and Application. 13th International Conference, pages 956959, Lisbon, Portugal, 2003.

[22] Ch. Clark and D. Schimmel; Scalable Pattern Matching for High-Speed Networks. In IEEE Symposium on FieldProgrammable Custom Computing Machines (FCCM), pages 249257, Napa, California, 2004.